

AD-A117 990

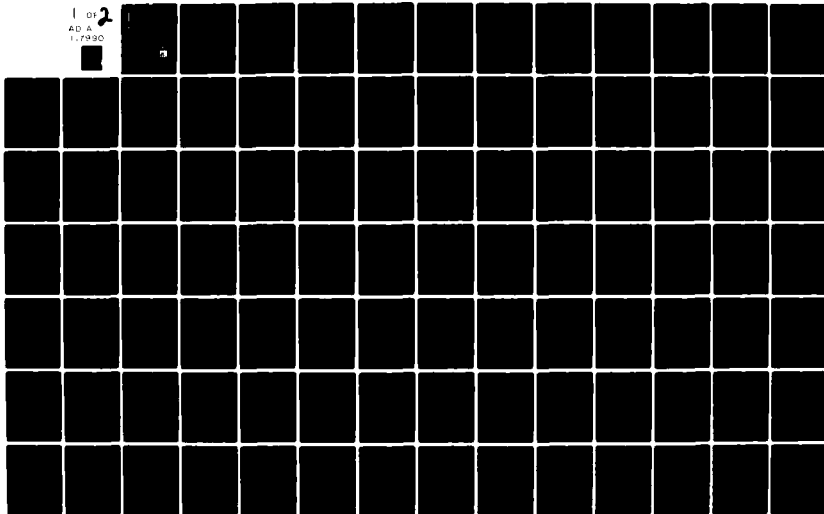
WEIZMANN INST OF SCIENCE REHOVOTH (ISRAEL) DEPT OF --ETC F/G 9/2
COMPARATIVE PREDICTION METHODOLOGY FOR DECENTRALIZED DDBMS.(U)
APR 82 M MELMAN AFOSR-81-0147

UNCLASSIFIED

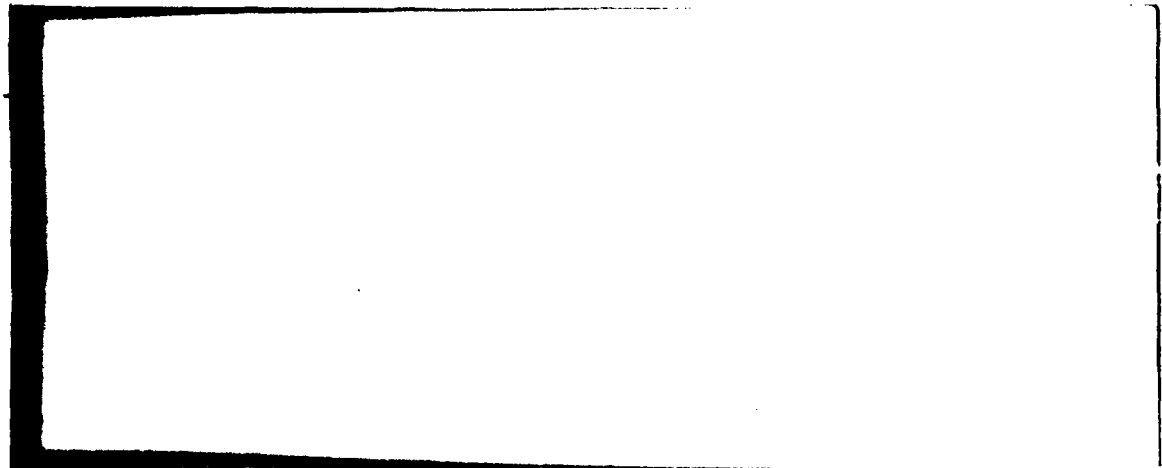
EOARD-TR-82-11

NL

1 of 2
AD A
1-7990



AD A117100



DTC FILE COPY

DTC
AUG 9 1982
H

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited



המחלקה למתמטיקה שמושית
DEPARTMENT OF APPLIED
MATHEMATICS

82 02 09 034

(2)

Grant Number: AFOSR 81-0147

COMPARATIVE PREDICTION METHODOLOGY FOR DECENTRALIZED DDBMS

Myron Melman
Department of Applied Mathematics
The Weizmann Institute of science
Rehovoth, Israel

30 April 1982

Final Scientific Report, 1 April 1981 - 31 March 1982

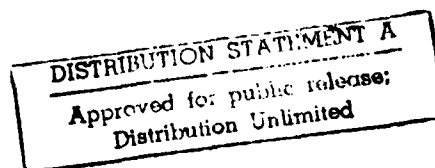
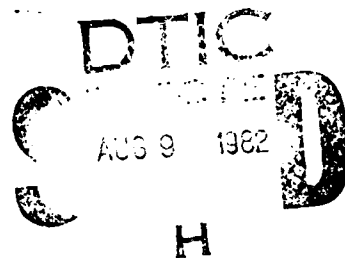
Approved for public release; distribution unlimited

Prepared for

UNITED STATES AIR FORCE
AIR FORCE OFFICE OF SCIENTIFIC RESEARCH
Building 410, Bolling AFB, D.C. 20332

and

EUROPEAN OFFICE OF AEROSPACE RESEARCH AND DEVELOPMENT
London, England



EOARD-TR-82-11


19 July 1982

This report has been reviewed by the EOARD Information Office and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

This technical report has been reviewed and is approved for publication.


JAMES R. KREER

Major, USAF
Chief, Information Sciences


ROSE RINGO
Scientific and Technical
Information Officer

FOR THE COMMANDER



PETER SOLIZ

Lt Colonel, USAF
Director, Aerospace Sciences

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. Report Number EOARD-TR-82-11	2. Govt Accession No. AD-A117 990	3. Recipient's Catalog Number
4. Title (and Subtitle) COMPARATIVE PREDICTION METHODOLOGY FOR DISTRIBUTED DDBMS	5. Type of Report & Period Covered SCIENTIFIC REPORT 1 APRIL 1981 To 31 MARCH 1982	
	6. Performing Org. Report Number	
7. Author(s) MYRON MELMAN	8. Contract or Grant Number AFOSR 81-0147	
9. Performing Organization Name and Address Department of Applied Mathematics The Weizmann Institute of Science Rehovoth, Israel 76100	10. Program Element, Project, Task Area & Work Unit Numbers	
11. Controlling Office Name and Address	12. Report Date 30 APRIL 1982	
	13. Number of Pages 75	
14. Monitoring Agency Name and Address	15.	
16. & 17. Distribution Statement Approved for public release; distribution unlimited.		
18. Supplementary Notes Appendix A is to be presented at the Summer Computer Simulation Conference, July 19-21, 1982, Denver, Colorado		
19. Key Words Simulation, Simulation Aids, Modular Structure, Distributed Systems, Fully Distributed Processing Systems, Processes, Nodes, Distributed Data Base, Distributed Multiple-Copy Data Base, Concurrency Control, Deadlocks.		
20. Abstract As distributed systems grow in size and complexity there is an ever increasing need to construct operating computer models of these systems prior to prototyping or actual system implementation. Simulation of Fully Distributed Processing Systems can entail a most serious modeling and simulation effort. This project deals with the modeling and simulation of Fully Distributed Processing Systems. Two areas of software development, a package directed as a simulation aid to support modularly designed system or nodal algorithmic structures and the model of a distributed multiple-copy data base concurrency control algorithm modularly designed to interface with the simulation support package are treated. The stand-alone Distributed System Simulator, (DISS), is presented in considerable detail to bring out its generality, flexibility and its applicability to a wide range of distributed systems. The algorithm of the SDD-1 Concurrency Control is presented along with simulation results.		

CONTENTS

<u>Chapter</u>	<u>page</u>
I. INTRODUCTION	1
II. THE DISTRIBUTED SYSTEM SIMULATOR	3
III. CONCURRENCY CONTROL ALGORITHM	6
Introduction	6
a complete system	6
System Model	8
Model for the SDD-1 Concurrency Control Algorithm	10
TM	10
External Alert Functions:	10
Internal Alert Functions:	11
Termination Alert Function	11
DM	11
External Alert Functions:	11
Internal Alert Functions:	12
Termination Alert Function	12
EM	12
IV. THE CONCURRENCY MONITOR	14
Structures Used by the Concurrency Monitor	14
Concurrency Monitor Priorities	15
Arrival of a READ	16
Arrival of a WRITE	17
Arrival of a NULLWRITE	18
Scanning the IP Messages	18
V. PERFORMANCE MEASUREMENTS	25
VI. PERFORMANCE EVALUATION	28
VII. CONCLUSIONS AND RECOMMENDATIONS	48
REFERENCES	49

Appendix

page

A.	THE DISS METHOD OF DISTRIBUTED SYSTEM SIMULATION	51
	ABSTRACT	51
	1. INTRODUCTION	52
	2. MOTIVATION FOR PACKAGE DESIGN	54
	3. THE STRUCTURE OF DISS	56
	4. NODAL COUPLING	58
	Active Transfer	59
	Passive Transfer	59
	5. THE PROCESS	60
	6. SIMULATION STUDY WITH DISS	64
	7. EXAMPLE	67
	Node Definition	69
	Internodal State Variables	70
	External Events	70
	Internal Events	71
	8. CONCLUSIONS	72
	REFERENCES	74
B.	SCAN - BLOCK DIAGRAM	75
C.	SIMULATION RESULTS	76



Accession For	
NTIS GRAY	
DTIC TAB	
Unannounced	
Justification	
By	
Distribution/	
Availability Codes	
Avail and/or	
Dist	Special
A	

Chapter I

INTRODUCTION

The Comparative Prediction Methodology for Decentralized DDBMS program, to be called 'effort' in this report, was undertaken to prepare a set of software programs that can serve multiple purposes. This report gives a status snapshot of that effort at the end of a 12 month period. Once the preliminary reviews of the scope of the program were completed it was obvious that the final objectives of the proposal could not be completely obtained in the one year that was allotted. However, it is the contention of the investigators that a considerable step toward the end goal has been attained.

As described in the proposal, two clearly defined and separate areas of interest were to be investigated, the development of a programming package that could serve as support and background for the simulation of distributed systems, and the development of a model for the simulation of the Concurrency Control Algorithm of the SDD-1 system. The two sections indeed run as a single program. However, the DIstributed System Simulator (DISS) package is a generalized support aid for the simulation of a wide range of distributed system models and is a stand alone support package. It is designed to work with any algorithmic description that converses satisfactorily with the DISS package. There are of course a set of rules that must be obeyed when designing the distributed system model so that the two major sections interface correctly and thus

mutually support each other. The second software section developed is a model of the SDD-1 Concurrency Control Algorithm. This model has been designed to operate with the DISS package and is now in operation. All programs were written in the Simscript II.5 programming language (Russ73).

This report contains a more detailed review of the two programming sections in Chapters II and III. The Concurrency Control Monitor designed for the Algorithm is discussed in chapter IV. A description of the Performance Measurements planned for the model and the Performance Evaluation is given in Chapters V and VI. The conclusions and recommendations are presented in Chapter VII.

Chapter II

THE DISTRIBUTED SYSTEM SIMULATOR

Repeated generation of simulation models representing queues and servers, computer systems, computer operating systems, and distributed systems has led to the concept of designing a generalized model that can be used as a basis by a large family of applications. The areas mentioned and especially the subject of distributed computer networks today offer sufficient common simulation requirements to warrant the formulation of a general framework within which one can modularly model a particular system.

The DIStributed Network Simulator, (DISS), package was designed to allow the modeler to concentrate his efforts on the unique algorithmic aspects of the system, and release him from preparing the standard repetitive elements of a simulation experiment of a distributed system model. Such an approach reduces the time of the model design period and of the amount of code required to prepare. At the same time this method offers a broad range of organization options and performance measurement features. A simulation experiment using DISS comprises two major sections when in execution, a) a modeler supplied set of processes and operating parameters and b) the fixed background framework.

The user supplied material describes the network characteristics such as configuration, operating conditions, node specifications and other applicable details that are peculiar to this model. The concept of a node is left to the modeler and can represent a simple control element or a complete independent system. The user supplies a process description of each unique type of node that is incorporated into the network. The processes are the modular elements of the system. The process contains all algorithmic descriptions of the tasks implemented by the node including the initialization of communications with other elements of the network. The user must also supply all the necessary parameters required for the simulation runs, and the output options. The processes that are prepared must contain the necessary interfaces to the background framework in order to establish the simulation structures required. These structures, based upon local nodal parameters, then belong to the particular node.

The background framework has been developed to meet the general simulation needs of a wide range of distributed system problems that can be represented as directed graphs. From the network input information the configuration and the nodal relationships are established. The nodes are initiated by the background routines that supply the necessary simulation structures required for the runs. These include the establishment of the measurement facilities offered by the programming language and requested by the individual node as required.

In addition to the standard automatic statistics collecting features of the language the user may call upon DISS routines that supply confidence intervals and autocorrelation data for selected variables. The package also contains a sophisticated tracing routine that permits a detailed step by step reporting of system conditions as specified by the modeler and is intended for debugging and verification purposes.

A more detailed and complete description of DISS appears in Appendix A as a self contained report. This report was submitted and will appear in the Proceedings of the Summer Computer Simulation Conference that is to take place on July 19-21, 1982 in Denver, Colorado. Reference material for the DISS package is found in Appendix A.

Chapter III

CONCURRENCY CONTROL ALGORITHM

3.1 INTRODUCTION

A distributed multiple copy data base management system represents a complex, highly sophisticated mechanism (Bern78, Bern79a, Bern79b, Bern80a, Bern80b, Roth80). The common denominator of all data base systems is that users enter queries in an attempt to retrieve information or to update the information maintained in the data base. If one is to model a specific feature of a certain type of system, it is necessary to analyze the system to determine those features that are common to the entire family of systems and the features that are unique to the particular system of interest. It is then sufficient to model only those features that are peculiar to the system of interest and to ignore the common operations. This chapter reviews those features of a distributed multiple copy data base system, selects those features that are directly involved with the concurrency control and finally presents the model for the simulation of the algorithm.

3.2 A COMPLETE SYSTEM

A generalized data base system can be characterized by the following sequential operations that are carried out to meet the needs of the user:

1. Arrival of queries

2. translation of queries to determine the exact requirement of the user
3. Determine the best location of the desired data and retrieve the data. This may involve large sorting operations depending upon the data base organization, the system architecture and the nature of the query.
4. Compilation of the query commands to perform the required operations on the retrieved data.
5. Execution of these operations at the locations of the retrieved data and then joining the partial results to form the final results.
6. Presentation of these final results to the user, if requested.
7. Updating the data base with the corrected data as requested.

If the data base is distributed over several sites, has multiple copies of the data base fragments, has multiple entry for queries and has no central control but has a local, cooperating, autonomous control for the execution of the queries at each site then the sequencing of the execution of the queries in order to maintain the consistency of the data base presents a serious problem. The basic requirement for global data consistency remains. This development program has been involved in the construction of a model of an algorithm that guarantees data base consistency when the data base is distributed with redundant copies of

selected fragments of the data base. These algorithms are called Concurrency Control Algorithms. Recently, two complete families of concurrency control algorithms, Two-Phase Locking (Man80), and Timestamp Ordering (Thom79), have been reported (Barn81).

This effort has concerned itself with the Conservative Timestamp Ordering algorithm associated with the SDD-1 program. In modeling such an algorithm for the type of data base required, the question is how much of the system operation is actually required to be modeled. Modeling an algorithm to determine its cost and efficiency can be interesting only if the performance results of the model may be compared with those of another competing algorithm running under similar conditions. The cost of implementing an algorithm is therefore reduced to all system delays that can be directly charged to the algorithm. In a data base system this can be interpreted as meaning the cost of the communications required by the algorithm and any additional delays determined by the implementation of the algorithm. But most important, it means that the cost of the data base operations common to all systems, mentioned in section 1 of this chapter may be ignored.

3.3 SYSTEM MODEL

The system model designed permits the modeler complete freedom in determining the system topology and the structure of the data base. The model as designed contains a system of 10 nodes, made up of five TMs and five DMs. The Data base is structured with 11 fragments randomly spread over the five DMs with a maximum of seven fragments at a single node. This permits for considerable replication of the fragments. Five tran-

saction classes are defined executing P1, P2 and P3. The five classes each have an allowable read-set and write-set consisting of selected fragments, in keeping with the conflicting class concept. When new transactions arrive at a class, each selects a random number of read and write fragments, up to the maximum number assigned to that class and then randomly selects the particular fragments. No duplications within a set are possible.

The transactions execute all of the possible communications as required by the SDD-1 system algorithm and these time delays are charged as part of the SDD-1 cost. Table 1 represents the message pattern of the system.

TABLE 1
System Messages

Msg.	From	To	Purpose
1	TM	Read DMs	READ
2	Read DMs	TM	READ Complete
3	TM	Read DMs	EXECUTE
4	Read DMs	FINAL DM	Partial Results
5	FINAL DM	TM	EXECUTE Complete
6	TM	FINAL DM	Start UPDATE Distribution
7	FINAL DM	UPDATE DM	UPDATE Distribution
8	Update DM	FINAL DM	Secure Memory Update Complete
9	FINAL DM	TM	Phase I Update Complete
10	TM	WRITE DMs	Phase II Write
11	TM	DM	Periodic NULLWRITE
12	DM	TM	Request for NULLWRITE
13	TM	DM	NULLWRITE in response to req.
14	Read DM	TM	READ Rejected

3.4 MODEL FOR THE SDD-1 CONCURRENCY CONTROL ALGORITHM

The model designed for the SDD-1 Concurrency Control Algorithm is based upon the use of the DISS package described in the previous chapter. In keeping with the DISS discipline three processes were designed, an Execution Manager (EM), a Transaction Manager (TM), and a Data Manager (DM). The TM and DM processes are fashioned after the suggested process design of the DISS package. As described in the DISS report in the Appendix, a nodal process is characterized by the External Alert Functions, the Internal Alert Functions and by the Termination Function executed by the process.

3.4.1 TM

The generalized TM Process executes the following functions:

3.4.1.1 External Alert Functions:

- | | |
|-------------------------------|-------------|
| 1. Read Complete | msg. no. 2 |
| 2. Transaction Rejection | msg. no. 14 |
| 3. End of Execution | msg. no. 5 |
| 4. End of Update Distribution | msg. no. 9 |
| 5. Request for a Nullwrite | msg. no. 12 |

3.4.1.2 Internal Alert Functions:

1. End of Message Transmission
2. Send Periodic Nullwrite msg. no. 11
3. Arrival of New Transaction

3.4.1.3 Termination Alert Function

3.4.2 DM

The generalized DM Process executes the following Functions:

3.4.2.1 External Alert Functions:

1. Arrival of Nullwrite msg. no. 13
2. Arrival of Write msg. no. 10
3. Arrival of Read msg. no. 1

Functions 1, 2, and 3 share in the use of the SCAN procedure.

The three functions and SCAN implement the Concurrency Control Algorithm.

4. Execute msg. no. 3
5. Partial Results Arr. Final DM msg. no. 4
6. Start Distribution msg. no. 6
7. Phase 1 Update msg. no. 7

8. End of Phase 1 Update

msg. no. 8

3.4.2.2 Internal Alert Functions:

1. End of Message Transmission
2. End of Access to Data Base
3. Timeout Request for Nullwrite

3.4.2.3 Termination Alert Function

3.4.3 EM

The Execution Manager process is designed to execute its required functions in a sequential manner so as to manage the complete experiment from start to finish. The following is a list of these functions:

1. Initialization and file control
2. Network Establishment
3. Network Topology printout
4. Experiment Control Parameter input
5. Data Base Definition
6. Node Establishment

7. Simulation Control

8. Schedule of Termination Function for all nodes

9. Log Printout

10. Release structures

11. End

Chapter IV

THE CONCURRENCY MONITOR

The order of accepting and processing requests for Data Access (DA) at any DM is assumed not to violate the rules mentioned in the SDD-1 references. A scheduling mechanism at each DM, called the Concurrency Monitor, is responsible for this ordering.

4.1 STRUCTURES USED BY THE CONCURRENCY MONITOR

In the SDD-1 system each READ message carries, among other things, a list of conflicting classes. The simulated model uses a global matrix to store conflicting classes instead of sending them with the read message. Each DM maintains a Concurrency Table (CT) ranked by class number. As can be seen from the Table 2, The CT contains two columns. The first column stores the timestamp of the most recently processed¹ WRITE request. The second column stores the timestamp of the most recently processed NULLWRITE message.

For each class I, a queue for messages called Concurrency Monitor Queue of I, CMQ(I), and a set for unmet read conditions called Conflicting Classes Read Conditions (CC.RC(I)) are created at each DM. Upon arrival, WRITE messages, NULLWRITE messages, and accepted READ messages are filed last in the corresponding CMQ. Messages are removed from the CMQs only at the moment they can access the local data base. The first

¹ Submitted to the data access queue but not necessarily completed.

TABLE 2

Concurrency Table

Class	Timestamp of most recently processed WRITE	Timestamp of most recently processed NULLWRITE
1	34527	34632
2	33674	33985
.	.	.
.	.	.

member in each CMQ is called Immediately Pending (IP). If the IP message in CMQ(I) is a READ with some unsatisfied read conditions, then, for each unmet read condition, a member identifying the class for which the condition is not met, is filed in the set of CC.RC(I). A member is removed from CC.RC(I) when the class it points too, fulfills the read condition.

- Access to the local database is through the Data Access (DA) queue which is a simple fifo queue.

4.1.1 Concurrency Monitor Priorities

Concurrency Monitor requests for data access is made at the DM by a low-priority-first mechanism. Each IP message is given a PR number which defines it's priority. The PR of a WRITE or a NULLWRITE is simply it's TS. For establishing the priority of an IP READ message, three cases are considered:

1. If all read conditions for the READ are met, PR is the read timestamp TSR.
2. Not all read conditions are met and no WRITE or NULLWRITE message is pending in the CMQ of any conflicting class which doesn't meet the read condition. In that case PR is also the TSR associated with the message.
3. Not all read conditions are met and at least one conflicting class with unmet read condition has a WRITE or NULLWRITE pending in it's CMQ. In this case the PR is the lowest TS among the WRITES or NULLWRITES pending in the CMQ's of the conflicting classes with unsatisfied read conditions.

Scheduling is done by lowest priority first except for one case which will be discussed below.

4.2 ARRIVAL OF A READ

An arriving READ message is first checked to see if it is immediately rejectable. A READ must be rejected only if a WRITE from a conflicting class with TS greater than the read timestamp TSR was already processed. The information in the CT is used for this check. In case of a rejection, a READ REJECTED message is sent to the supervising TM. The TM retries the Transaction READ operation up to 10 times before abandoning the Transaction. If READs are distributed among several DMs, those READs that are not rejected must be cancelled. The TM assigns a new TS for each new retry until the cause of the rejection is eliminated. An

accepted READ is filed last in the corresponding CMQ and if it is IP, read conditions are checked, PR is calculated and a SCAN procedure, discussed later in this section, is called.

4.3 ARRIVAL OF A WRITE

WRITE messages are never rejected. A new WRITE from class I called $W_i(I)$ has just ARRIVED. If no other WRITE or NULLWRITE precedes $W_i(I)$ in $CMQ(I)$ then the read condition and the PR for each IP READ with class I as an unsatisfied read condition is checked. Assume that I is in J's set of conflicting classes and there is an IP READ called $R_j(J)$ in $CMQ(J)$. The PR of $R_j(J)$ is PR_j and its TSR is TSR_j . The TS of $W_i(I)$ is TS_i . Three cases are considered:

1. $TS_i < PR_j$. The new WRITE has a lower timestamp than the one that previously determined, PR_j . PR_j is set to TS_i and the read conditions are not changed.
2. $PR_j < TS_i < TSR_j$. No correction is needed. $W_i(I)$ does not fulfill the read condition of $R_j(J)$ and TS_i is not the lowest timestamp in the unsatisfied read conditions of $R_j(J)$.
3. $TSR_j < TS_i$. $W_i(I)$ satisfies the condition of $R_j(J)$. The member specifying I as an unmet condition is removed from $CC.RC(J)$.

If a modification is made to any IP READ or if $W_i(I)$ is IP in $CMQ(I)$, the SCAN procedure is called.

4.4 ARRIVAL OF A NULLWRITE

Since more than one NULLWRITE can carry the same timestamp, new NULLWRITES are ignored if a NULLWRITE with the same timestamp already exists or was processed. Otherwise, the same procedure as for a new WRITE, is performed for the new NULLWRITE.

4.5 SCANNING THE IP MESSAGES

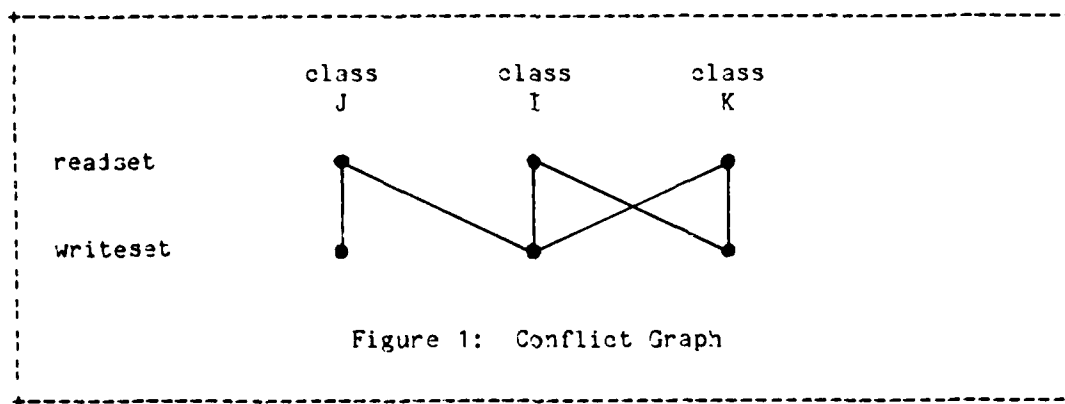
Whenever a change is made to an IP message, the SCAN procedure is called. SCAN is responsible for transferring DA requests from the CMQs of a particular DM to the DM's DA queue. The transfer is made as soon as possible and without indefinitely postponing any request (Bern80b).

As a first step, IP messages at the DM are scanned for the minimum priority, MIN.PR. No two IP WRITES or two IP NULLWRITES or IP WRITE and IP NULLWRITE can have the same PR. This is true because their PRs are equal to their TSs and messages from different classes can not have the same TS (a WRITE and a NULLWRITE from the same class can have the same TS but can't be both IP). The problem is that there can be a single IP WRITE or IP NULLWRITE and one or more IP READs, all having the same MIN.PR. This situation can occur when for all those READs the WRITE (or NULLWRITE) is the unsatisfied read condition with the lowest timestamp. If this is the case, the WRITE (or NULLWRITE) is taken as the MIN.PR. When more than one message has the same MIN.PR, but all of them are IP READs, one of them is taken as the MIN.PR.

A simple low priority first mechanism has some potential deadlocks. One of the potential deadlocks is discussed in (McLe81). As an example

of a situation that can cause that kind of deadlock, consider a system with two classes, I and J. I is in conflict with J and J is in conflict with I. There are only two transactions in the system, i in I and j in J. Each transaction sends it's READ to a different DM and waits there for the other class to satisfy the read condition. Since no transaction can process and no DM can detect the deadlock, this deadlock can't be broken unless NULLWRITES are sent. In the simulated algorithm a READ that has reached the point of MIN.PR and can not fulfill all of it's read conditions for a timeout period causes the request of a NULLWRITE message for each unmet read condition.

A second potential deadlock is described in (Bern90b) It arises in the above situation when the READs arrive at the same DM. It can easily be seen that no special action must be taken in addition to NULLWRITE request. The deadlock breaking described in (Bern90b) is not needed and will not help solve the problem in the system considered here.



The third type of deadlock actually occurred in the model discussed here. As an example consider a system with three classes of transactions, I, J and K. The conflict graph caused by their read and write sets is shown in Figure 1. As can be seen from this graph, class J runs P1 against class I and classes I and K run P3 against each other. Only two transactions exist in the system, i in I and j in J. Their timestamp order is $TS_j > TS_Rj > TS_i = TS_{Ri}$. The read messages, R_i and R_j , are sent to the same DM. R_j arrives first and requests a NULLWRITE from TM_i (the TM supervising class I). As R_i arrives it becomes the MIN.PR ($PRI = TS_i$) and requests a NULLWRITE from TM_k . Meanwhile the NULLWRITE message from TM_i arrives carrying a timestamp which is equal to TS_i . R_i and R_j have the same priority. Assume R_j is selected to be the MIN.PR message. R_j will remain as the MIN.PR until the TM_i NULLWRITE will be processed. The TM_i NULLWRITE cannot be processed until R_i will be processed and R_i cannot be processed because it is not the MIN.PR message. A deadlock situation exists.

The next example of the third type of deadlock is more general. Let class I be in the set of conflicting classes of J, so J must perform P1 against I. Three transactions are considered, i and i' in class I and j in J. Their timestamp order is $TS_i < TS_j < TS_{i'}$ and for simplicity assume that for each one of them $TS_R = TS$. Assume that i arrived first in the system and it's R_i (read message) found an idling DM. Clearly i can accomplish it's read and execution phases. Meanwhile, j enters the system and it's R_j reaches some DM, say DM_k . R_j can not be processed because it's read condition I is not satisfied. Transaction i' presents itself at $TM(I)$ before i has finished its execution phase. If $R_{i'}$

doesn't conflict with W_i , $TM(I)$ can send R_i' without waiting for the completion of i . Assume that R_i' is sent to DM_k . R_i' is IP and has no read condition to meet, so, it could have read but it is not the MIN.PR message at DM_k . Therefore R_i' will wait until R_j will be removed from the CMQ. Transaction i now sends W_i to DM_k where W_i is filed after R_i' in $CMQ(I)$. As a result, the priority PR_j of R_j is lowered to TS_i so R_j still has the MIN.PR. After a timeout period R_j will request a NULLWRITE from $TM(I)$. The latest timestamp this NULLWRITE can carry back is TS_i' which is large enough to satisfy the read condition of R_j' . Unfortunately W_i is still pending in the $CMQ(I)$ and filing the NULLWRITE after W_i will not do any good. On the other hand, moving it to the head of $CMQ(I)$ as suggested in (Bern80b) will violate the write pipelining rule, therefore it is forbidden.

To avoid this kind of deadlock, one more scheduling rule is needed: If the IP message with the minimum priority MIN.PR is a READ (R_j) whose lowest priority unsatisfied read condition is a message from class I, M_i , with TS_i , then, M_i is moved to the head of $CMQ(I)$, and processed immediately. According to the rules for calculating the PR of a READ, M_i can be a WRITE or a NULLWRITE but not a READ message. Clearly M_i is not IP since its priority is equal to the MIN.PR but it was not selected as the MIN.PR message. It is also easily seen that only READs can be ahead of M_i in $CMQ(I)$. A WRITE (or NULLWRITE) preceding M_i in $CMQ(I)$ must have a lower PR contradicting the choice of M_i as the lowest priority unsatisfied read condition. If M_i is a NULLWRITE, processing it before its preceding READs will not effect those READs because M_i will not make any change in the data base files. If M_i is a WRITE, the

READs preceding it belong to transactions that will write after M_i , therefore they must have later timestamps. Those READs would not have been released from $TM(I)$ unless they didn't conflict with i 's WRITE message M_i . So, processing M_i first will not change data items needed for those READs.

Accompanied by the previous rule the correctness of the lowest-priority-first scheduler can be shown. Let M be the message with the lowest priority. If M is a NULLWRITE it can be processed immediately. NULLWRITES are not held by READs and must obey only the write pipelining rule. If M is a WRITE it will be held only by an IP READ with a read condition that has a timestamp smaller than M , contradicting the choice of M . Therefore the WRITE can be immediately processed (Bern80b). If M is a READ, three situations must be checked:

1. All read conditions are met: The READ can immediately be processed.
2. Not all read conditions are met and there exists a pending message called M_i which is M 's unmet read condition with the lowest timestamp. This is the situation in which the extra rule is invoked and M_i will be passed to the head of $CMQ(I)$ and become the MIN.PR.
3. Not all read conditions are met but no pending WRITE or NULLWRITE exist in any CMQ of any unmet read condition. For each unmet read condition a WRITE or NULLWRITE will eventually arrive. Assume M_i arrived. If I is the last unmet read condition in M

and M_i satisfies this condition, case 1 exists. If M_i satisfies I 's condition but M still has other unmet read conditions, this case remains. If it does not satisfy I 's read condition case two exists. Since there are only a finite number of timestamps smaller than TSR of M , eventually case one will exist.

After each pass of SCAN, a message is chosen. This message is called M . M can be a READ, WRITE or NULLWRITE and for each type different actions are taken.

1. M is a WRITE: M can be immediately transferred from the CMQ to the DA queue. In addition the following operations are performed.
 - a) The TS of M must replace the most recently processed WRITE timestamp stored in the Concurrency Table (CT) in the row corresponding to M 's class.
 - b) For each IP READ which M is its lowest priority unsatisfied read conditions, recalculate the PR and check the read conditions after removing M from its CMQ.
 - c) If after removing M from its CMQ the IP message in that CMQ is a READ, calculate the PR and set the read conditions for that READ.
2. M is a NULLWRITE: The NULLWRITE is treated as a WRITE with two exceptions. First, the NULLWRITE is destroyed after being removed

from the CMQ since it doesn't access the data. Second, it's TS is stored in the most recently processed NULLWRITE column in the CT.

3. M is a READ: If not all read conditions for M are satisfied M will remain in the CMQ and the scheduler will wait for further information. Otherwise, the READ can access the data. It will be removed from the CMQ and filed in the DA queue. In addition, if the IP message, after removing M from the CMQ, is a READ, it's PR is calculated and it's read conditions are set.

If any modification was made during the SCAN pass, another pass of SCAN is called.

The DA queue is a FIFO queue with a single server. The average time needed for one access is an internal parameter at each DM and it can be fixed through the input information of the DM. Access time is exponentially distributed around this average. When a READ message ends data access, a READ-ENDED message is sent to the supervising TM. If a WRITE ends its data access a check is made to see if this is the last WRITE message for the transaction, and if so, the transaction is destroyed.

A block diagram describing the SCAN procedure is given in Appendix(B).

Chapter V

PERFORMANCE MEASUREMENTS

The model developed for the SDD-1 Concurrency Control Algorithm charges time delays in the simulation model to the following operations:

1. Time lost due to queusing for execution at arrival.
2. Message transmission time (when messages are sent in parallel, the longest time interval is considered).
3. Queusing time in the Concurrency Monitor while a Read or write request is waiting to be released for data access.

No additional overhead or operational time charges are made in the formation of the response time of a Transaction.

The performance measurements are to be presented as a series of graphs individually representing behaviour of Transactions running under P1, P2 and P3 respectively. The behaviour of each protocol type is again separated over a series of Transaction Interarrival Time graphs. The arrival rates are chosen to demonstrate the system response at the limits of a lightly loaded system, a near saturated system and a set of mid-range arrival rates. The variables measured are

1. Transaction Response Time - the time interval beginning with the transaction arrival time and terminating when the TM sends the

write message to the DM. It is assumed that all transaction read and write.

2. The number of Transaction rejections. (Rejections are retried 10 times and are then retracted from the system).
3. Queueing time in the Concurrency Monitor - when a read or write request is waiting to be released for data access. (When parallel efforts are made, the longest queueing time is used).

These variables are plotted as a function of the delta-read-time used to form the Read Timestamp for the P1 and P2 type transactions. Delta ranges from 0.01 to 0.6 of the interval between this timestamp and the timestamp of the previous transaction of that class. Each graph contains a family of curves where the curve parameter is the NULLWRITE timeout interval.

Due to insufficient time to complete the measurements of the model as described in this report, a set of measurements similar to those described above but relating to a simplified model as presented here. The simplification of the measured model lies in the assumptions made for the data base. In the simple data base there are no fragments. The data base as a unit is replicated at each DM. Such a configured data base then reduces the number of messages sent. In addition the simple model has no facilities to retry transactions that were rejected during the attempt to read. Such transactions are abandoned in the simple model. All other rules for the implementation of the transaction classes, protocols, and the concurrency control algorithm are the same.

Appendix C contains two sample printouts of short simulation runs of the simplified model. The first set of output data represents a run made for the collection of statistics and so contains no additional tables. The second run was made to illustrate the tracing capability of the DISS package, the Transaction Completion Table and the DM LOG as prepared by the SDD-1 algorithm processes.

Chapter VI

PERFORMANCE EVALUATION

This chapter presents the performance measurements made on the SDD-1 model representing the simplified system and thus the simplified data base.

The simulated DB is a collection of replicated logical fragments. Four classes of transactions, denoted as class #1 through #4, are defined. Class definitions and the corresponding conflict graph are presented in Figure 2. As can be seen from the conflict graph, classes 1 and 4 have no conflicting classes. Class 2 must perform P2 against classes 1 and 3, and class 3 runs P3 against class 4.

The network on which this DB is implemented, is composed of two centers with three nodes in each one of them. Figure 3 shows the network topology. The number inside each node is the node ID number, and the number on every communication line is the delay, in milliseconds, introduced by the line when a message is sent through it. As can be seen, the communication between nodes from different centers is twenty times more expensive than the communication within the center. Nodes 1 through 4 are TMs supervising the corresponding classes. Nodes 5 and 6 are DMs.

The effects of three parameters on the system performance is described. The first parameter, called Delta, is the portion of the

Class Definitions

class	1	2	3	4
readset	u	v,w,x	y	z
writeset	u,v	x	w,y	y,z

The Corresponding Conflict Graph

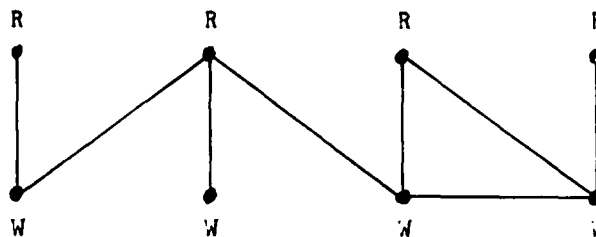


Figure 2: Class Definitions and Conflict Graph

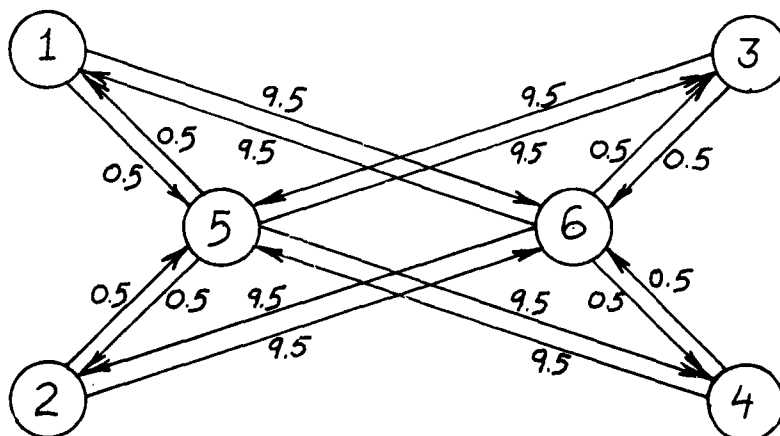


Figure 3: Network Topology

'permitted range' subtracted from the transaction timestamps to create the read timestamps. The second parameter is the interarrival time, IAT, of new transactions at the TMs. And the third is the timeout interval, TO, invoked before requesting or sending a NULLWRITE. The system performance is characterized by the average Response Time, the average time spent by READ requests at the CMQs (Read CMQ Time), the proportional number of rejected transactions, and the average number of messages sent per transaction. The simulation results are summarized in the following graphs. All time units are in milliseconds.

Graphs 1 through 7 describes the average response time as a function of Delta, for different classes at different timeout values, TO, for different interarrival times, IAT. These graphs show a variation of the response time of class 3 as a function of Delta. Class 3 runs P3 therefore it is not directly influenced by the value of Delta. The variation in response time shows the dependency between classes in this system. The most sensitive class, as can be seen from these graphs, is class 2. Therefore the remainder of this chapter will concentrate on this class.

Graphs 8,9 and 10 present the average READ CMQ time of transactions from class 2 as a function of Delta. The TO and IAT are similar to those in graphs 5,6 and 7. Comparing the response time with the read CMQ time shows a similar behaviour when the system is not loaded (large delta). When the load increases and the response time climbs very rapidly, the read CMQ time tends to stabilize on a value almost independent of the IAT. This value indicates an approximate maximum rate, under a given TO, at which the DM can handle data access requests.

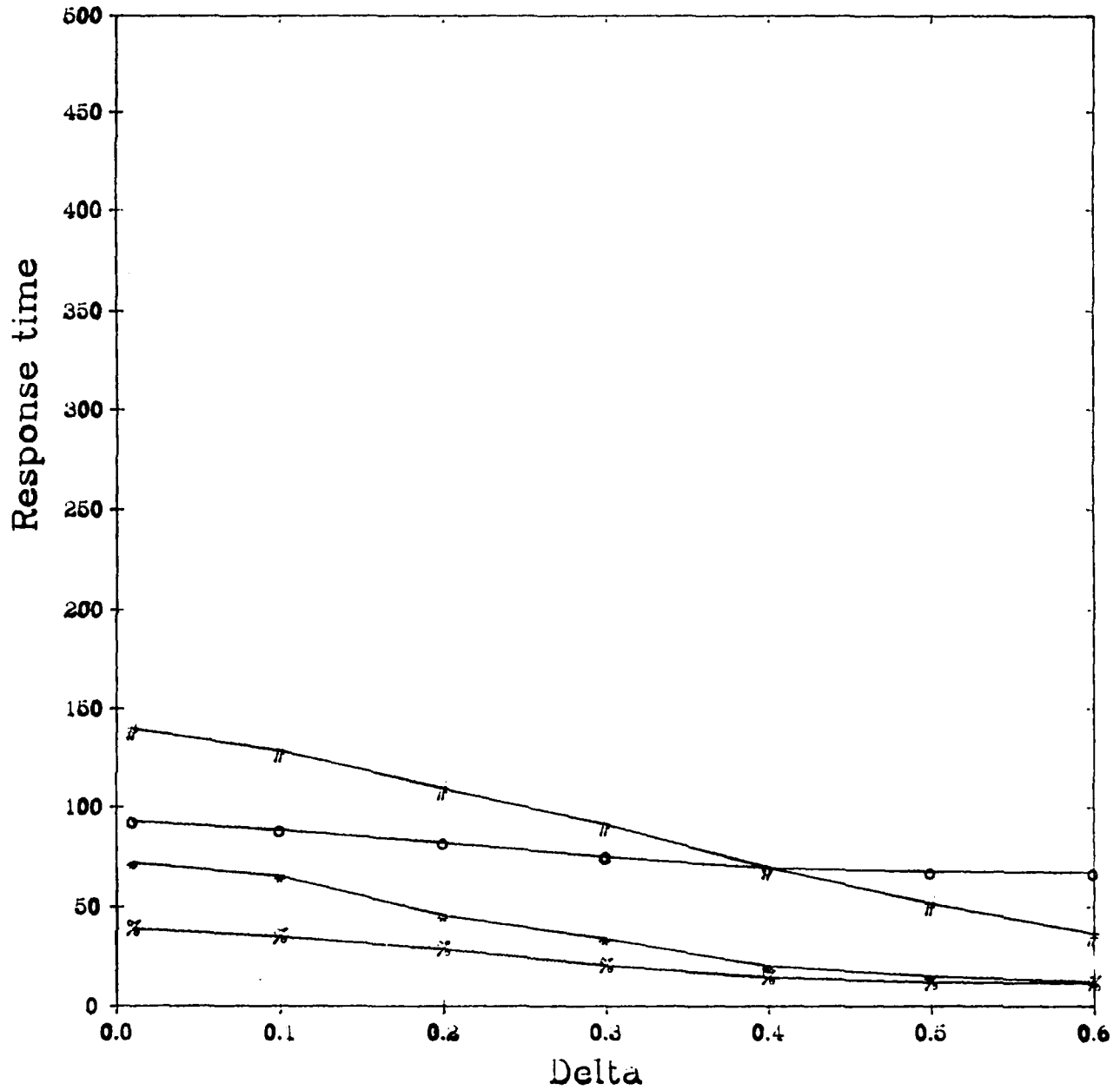
A trade off can be seen by comparing the response time behaviour (graphs 5,6,7) and the amount of transaction rejections (graphs 11,12,13) as a function of Delta. Increasing Delta, improves the response time but increases the number of rejections. A compromise will be found. In the given example, a Delta of 0.2 - 0.3 seems to be the best choice.

The response time, the read CMQ time and the number of rejections, are improved as T0 is decreased. On the other hand, the number of messages sent per transaction, increases as T0 gets smaller (graphs 14,15,16). In a system where the communication lines are used not only for the DDBMS, and the lines are loaded, increasing the number of messages is not desirable. Graphs 14,15 and 16 indicates that for this system, the number of messages increases very sharply for T0 below 70 milliseconds. Taking into account all these effects, the measured system is expected to give good results using Delta= 0.2 - 0.3 , and T0= 70 - 100 milliseconds.

Graph 1 :

RESPONSE TIME OF ALL CLASSES

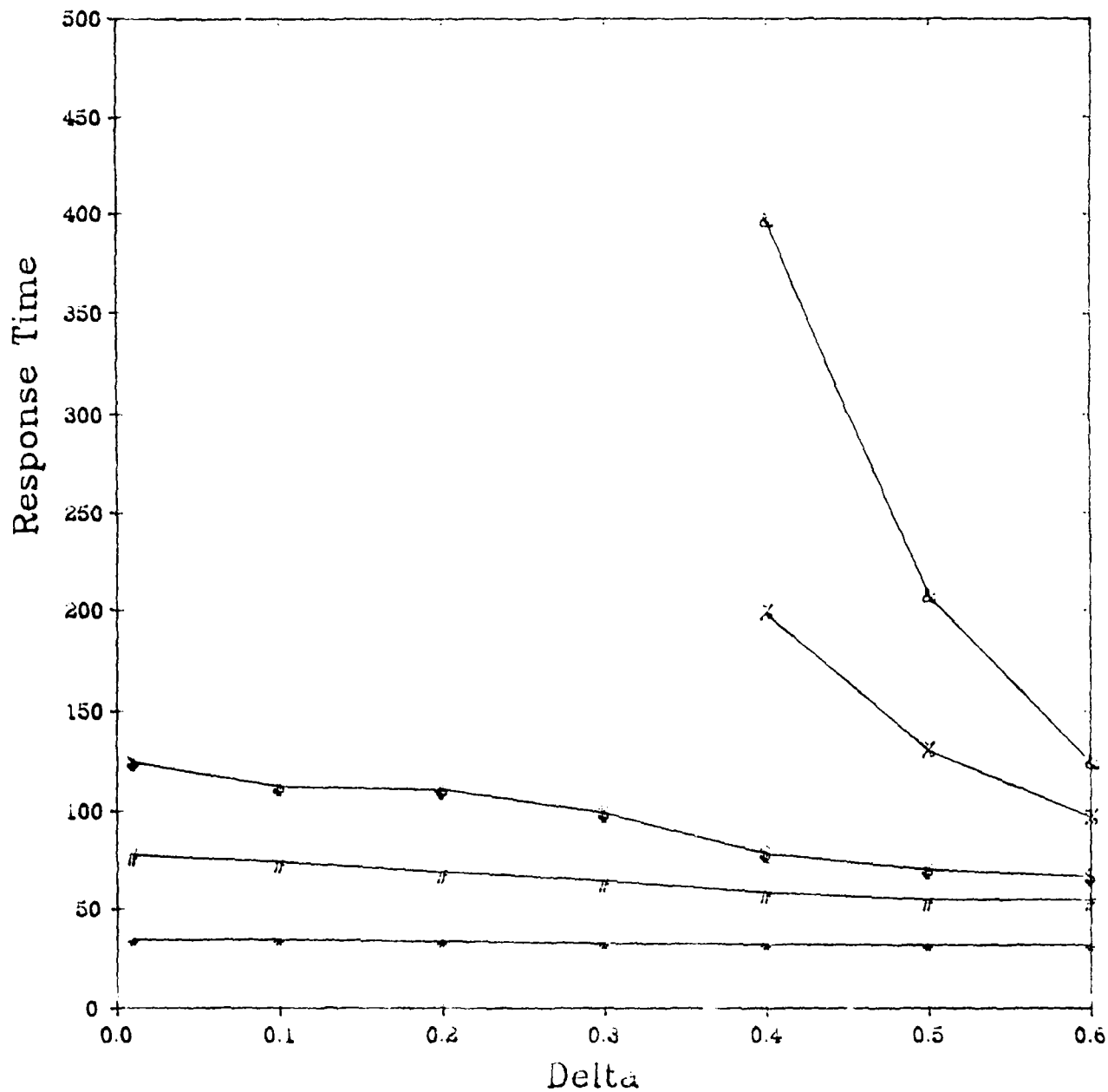
IAT=100 TO=100 CLASS // 1--* : 2--// : 3 -o : 4-%



Graph 2

RESPONSE TIME OF CLASS #3

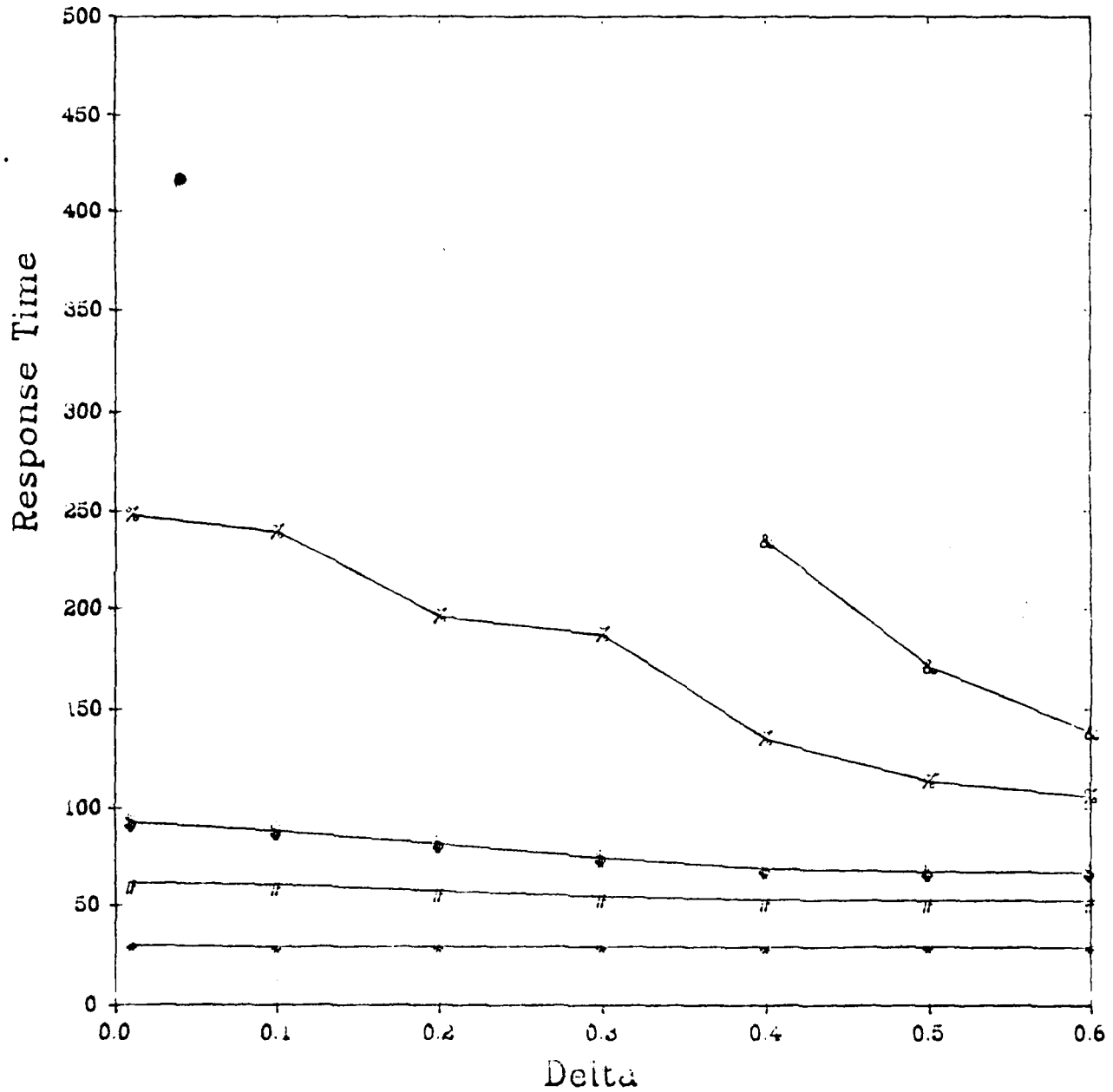
LAT = 60 TO = 30 - * 70 - # 100 - \$ 200 - % 300 - &



Graph 3

RESPONSE TIME OF CLASS #3

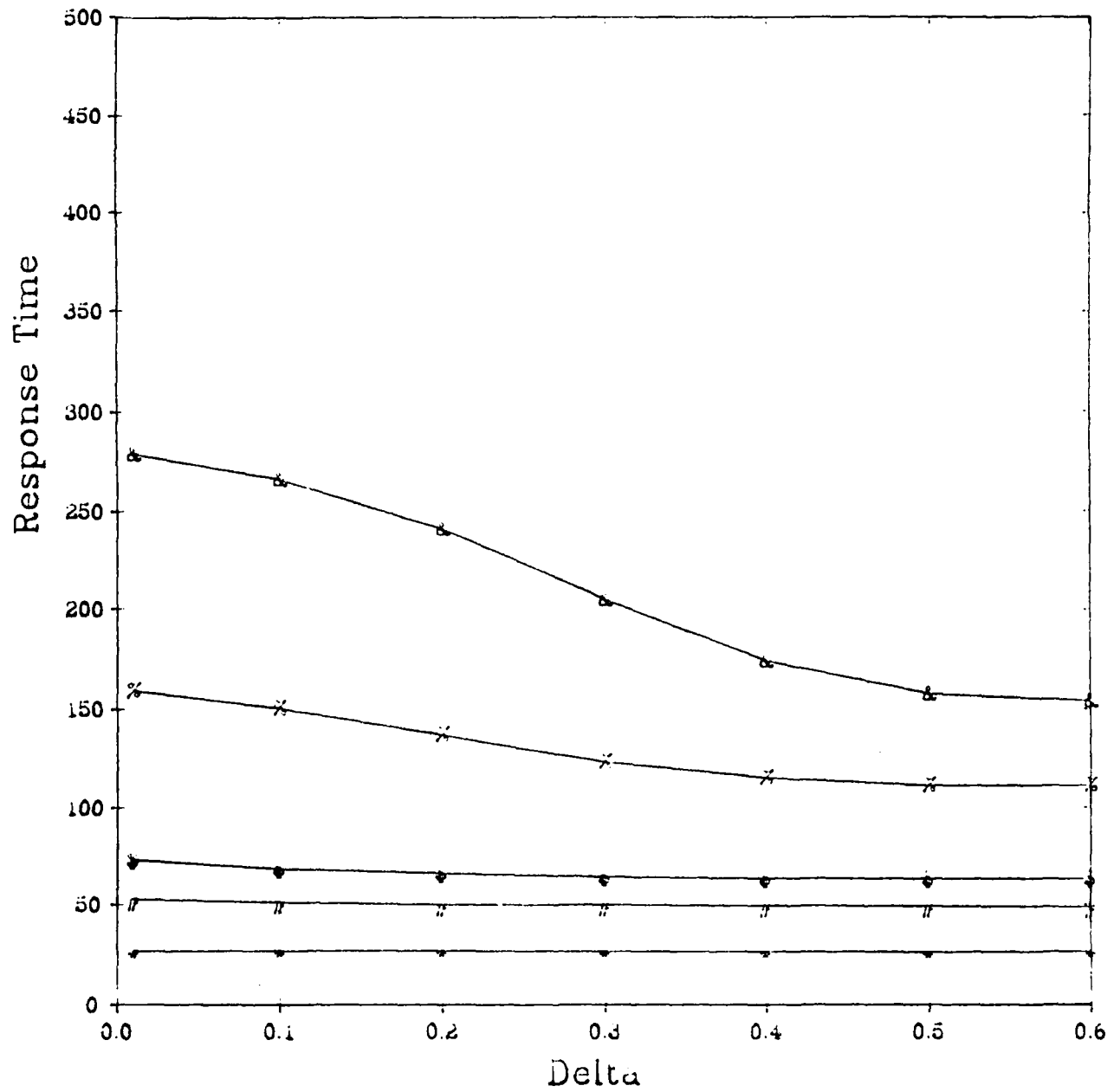
LAT=100 TO = 30-* : 70-# : 100-\$: 200-% : 300-&



Graph 4

RESPONSE TIME OF CLASS #3

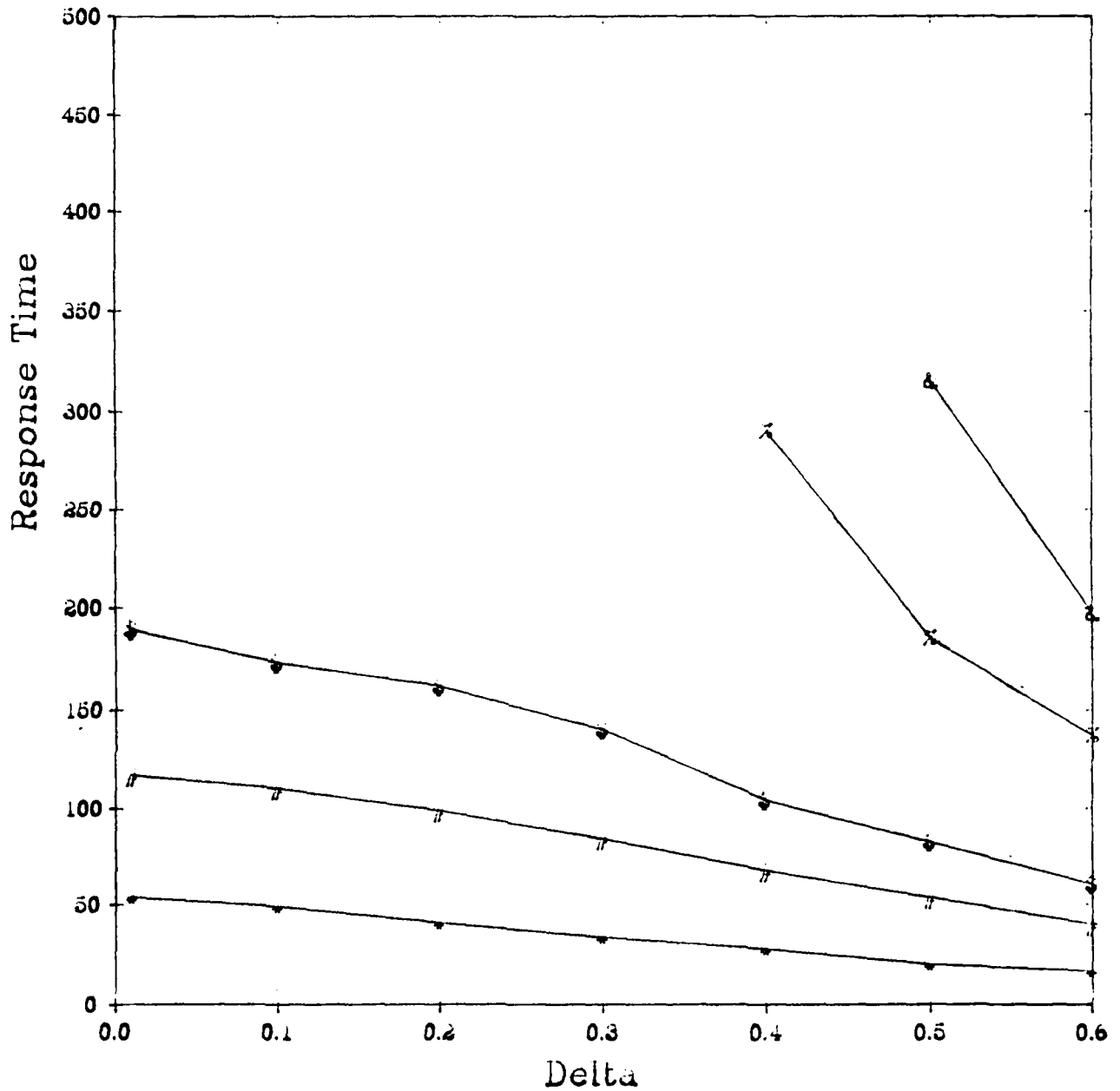
LAT=200 TO = 30 - * 70 - # 100 - \$ 200 - % 300 - &



Graph 5 :

RESPONSE TIME OF CLASS //2

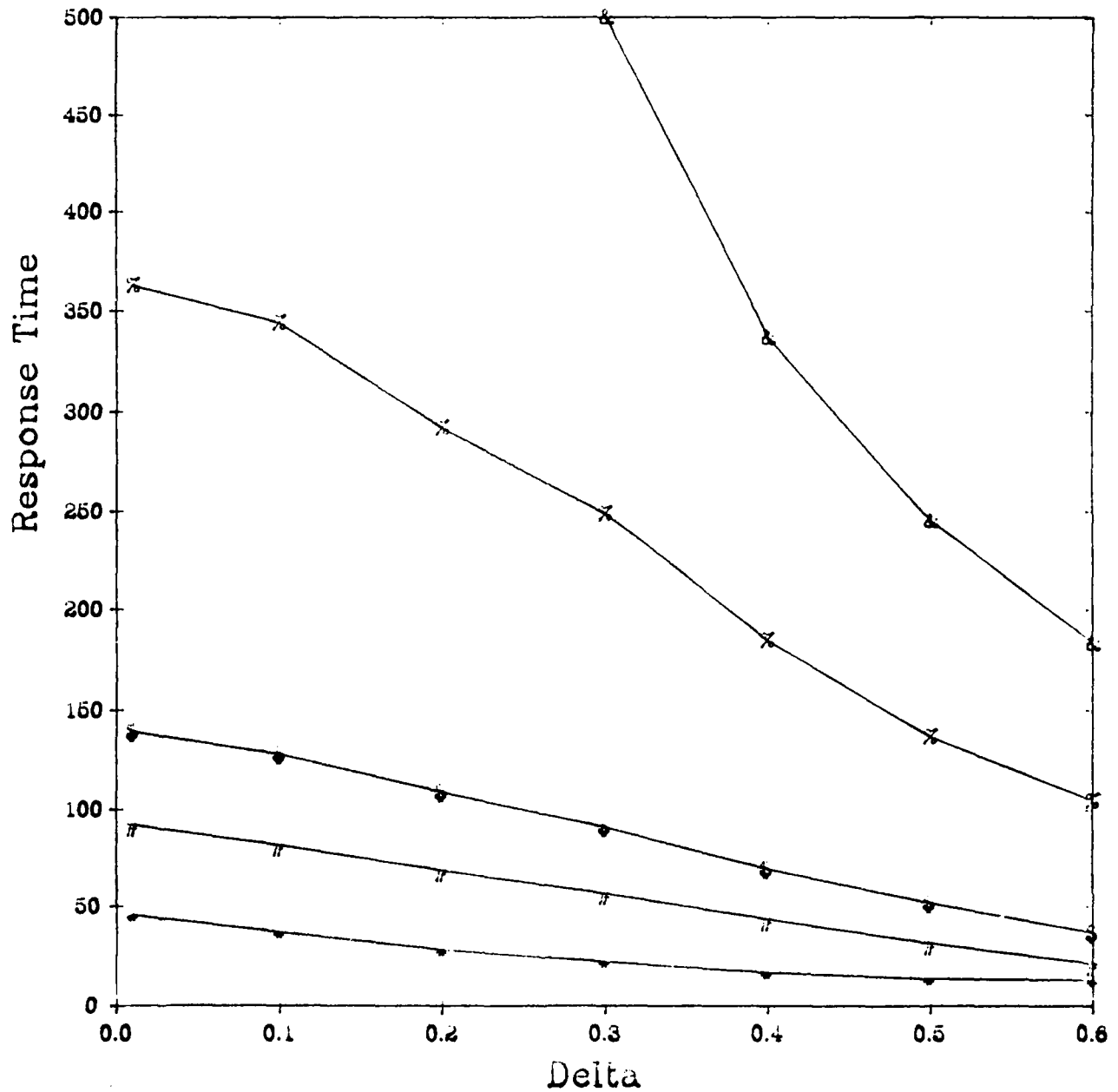
LAT= 60 TO = 30-+ ; 70-// ; 100-\$; 200-% ; 300-&



Graph 6 :

RESPONSE TIME OF CLASS #2

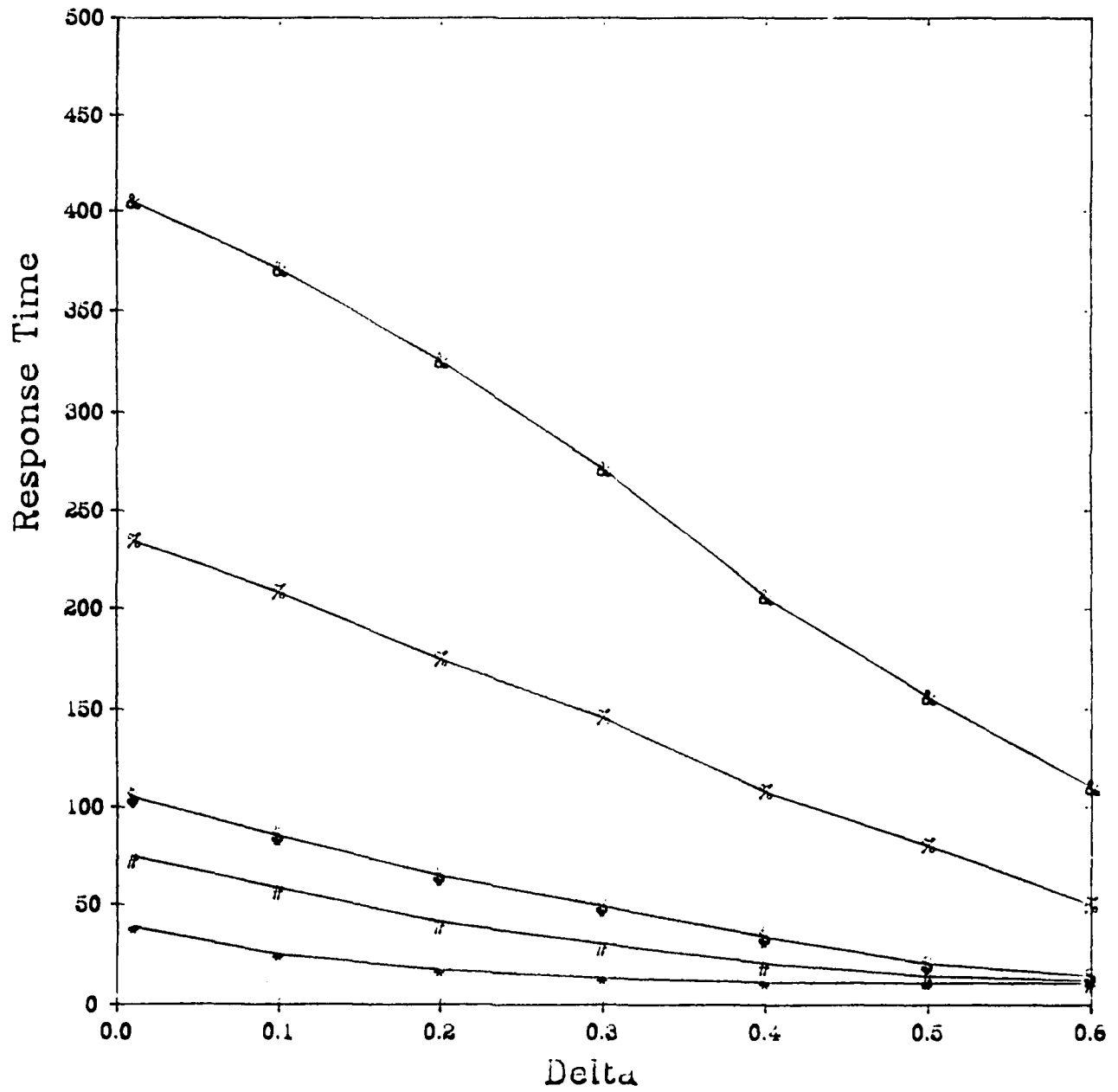
LAT=100 TO = 30-* ; 70-# ; 100-\$; 200-% ; 300-&



Graph 7 :

RESPONSE TIME OF CLASS #2

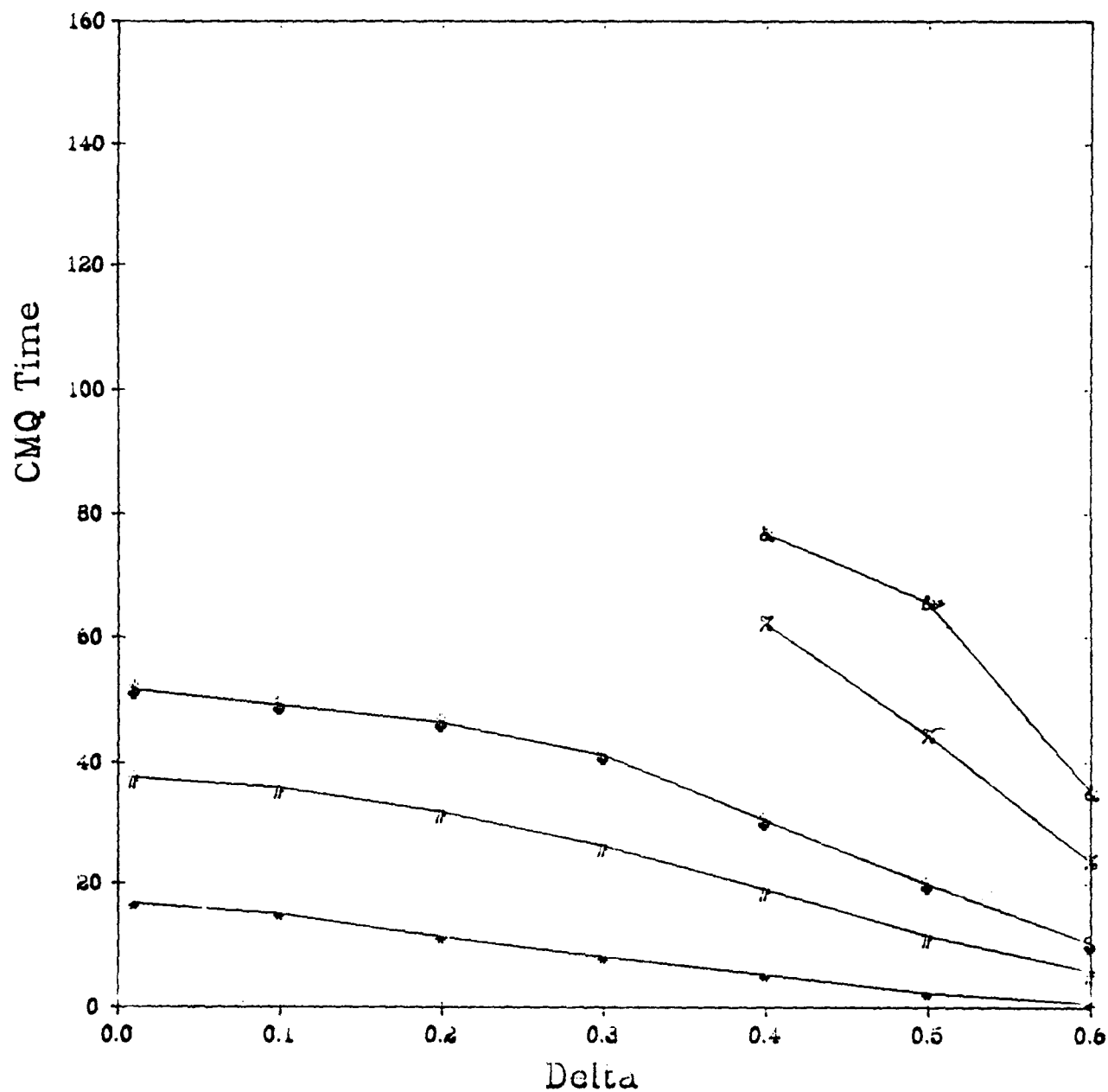
LAT=200 TO = 30-* ; 70-# ; 100-\$; 200-% ; 300-&



Graph 8 :

READ MESSAGES CMQ TIME AT DM #5

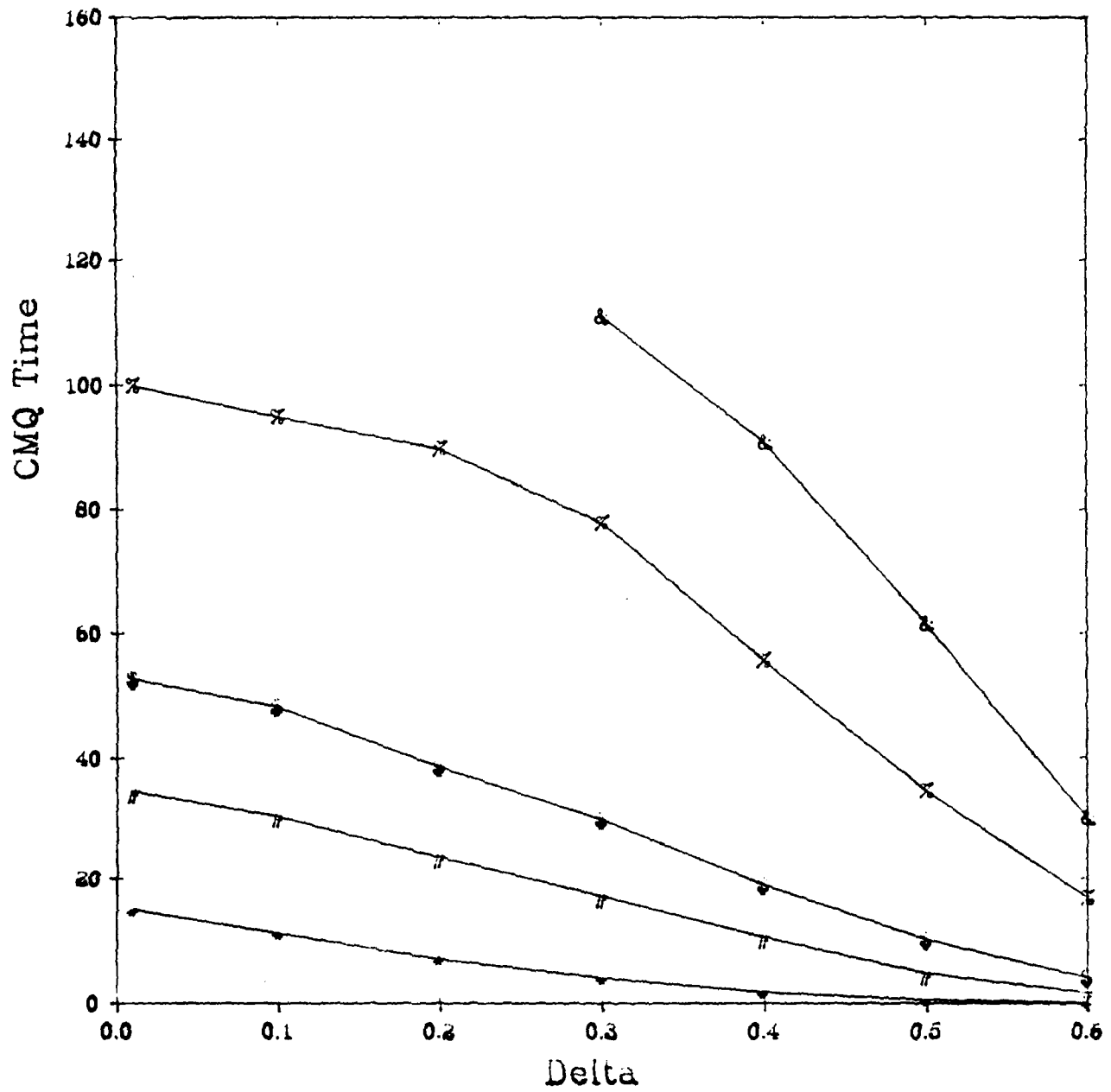
LAT= 60 TO = 30-* : 70-# : 100-@ : 200-% : 300-&



Graph 9 :

READ MESSAGES CMQ TIME AT DM #5

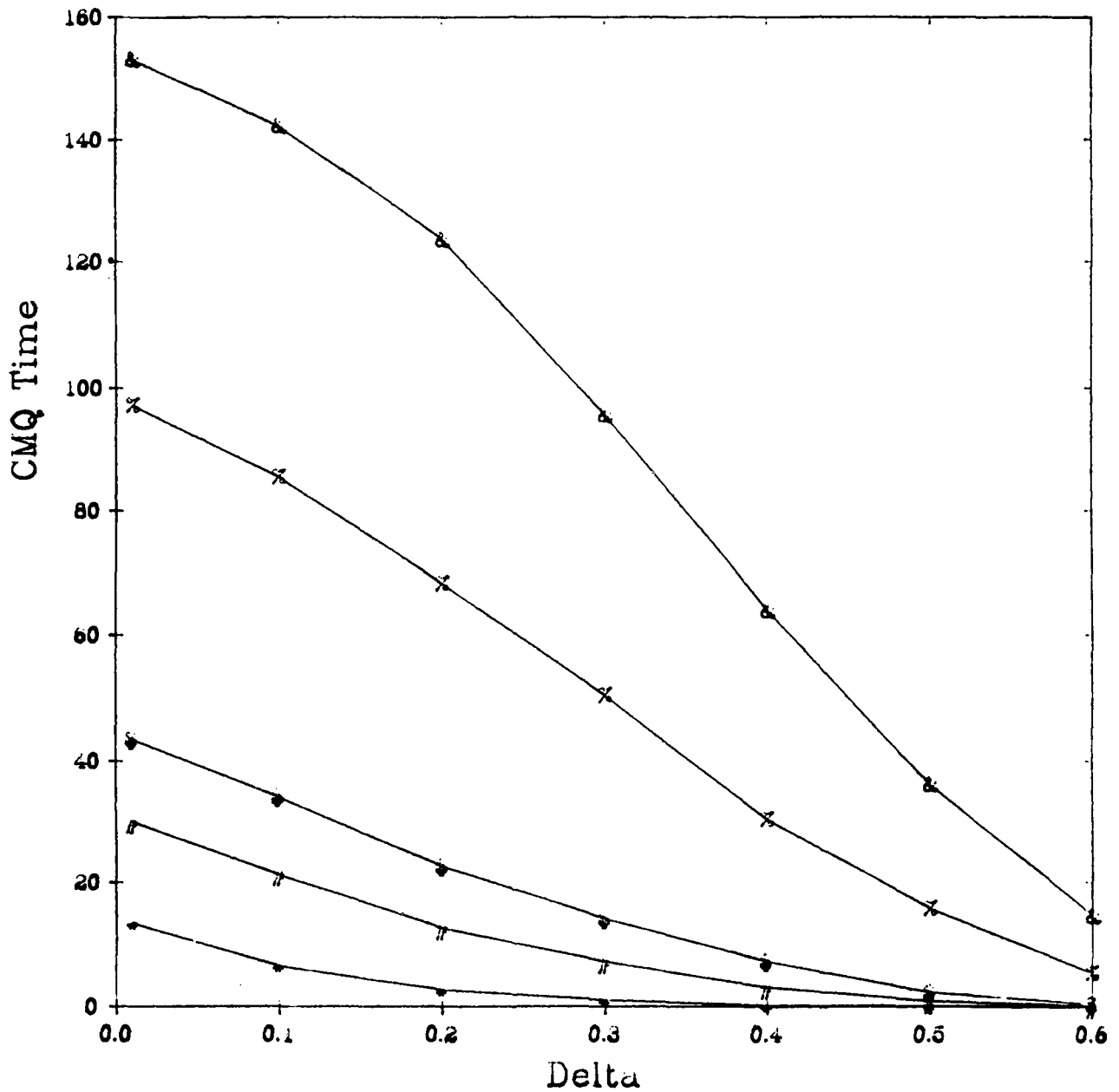
LAT=100 TO = 30-+ ; 70-# ; 100-§ ; 200-% ; 300-&



Graph 10 :

READ MESSAGES CMQ TIME AT DM #5

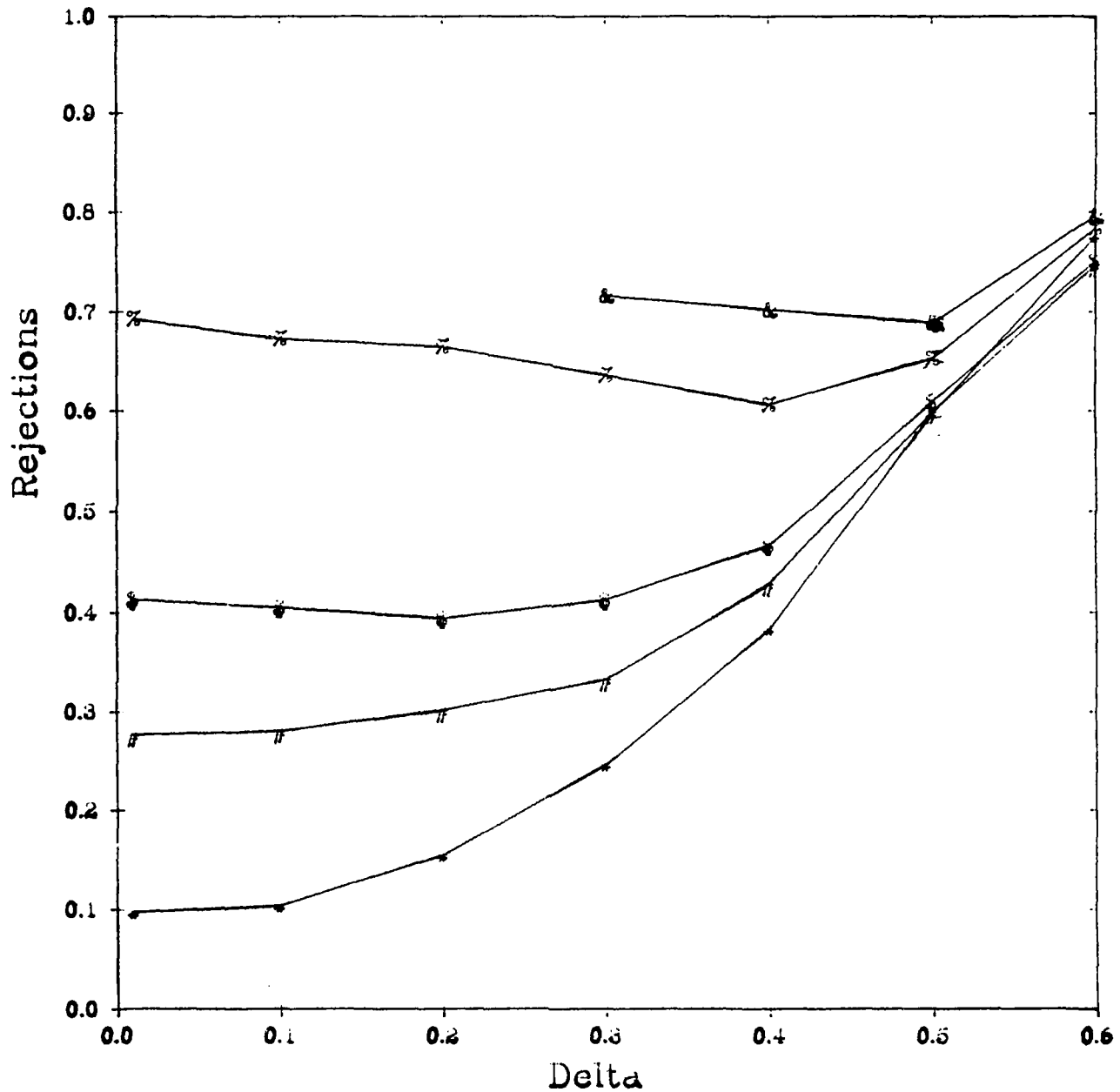
LAT=200 TO = 30-+ ; 70-# ; 100-\$; 200-% ; 300-&



Graph 11 :

REJECTIONS OF TRANSACTIONS FROM CLASS #2

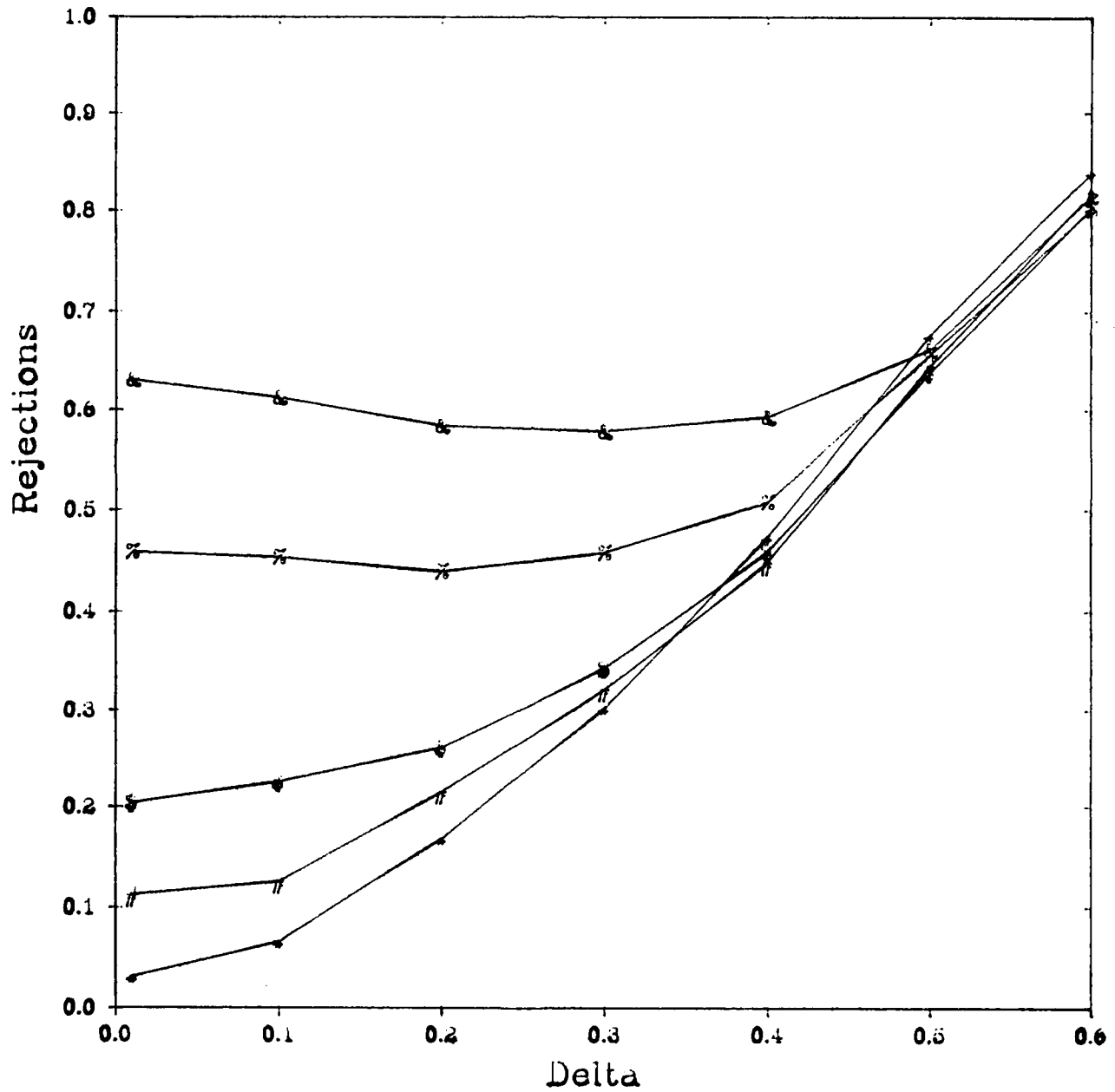
LAT= 60 TO = 30-* : 70-# : 100-\$: 200-% : 300-&



Graph 12 :

REJECTIONS OF TRANSACTIONS FROM CLASS # 2

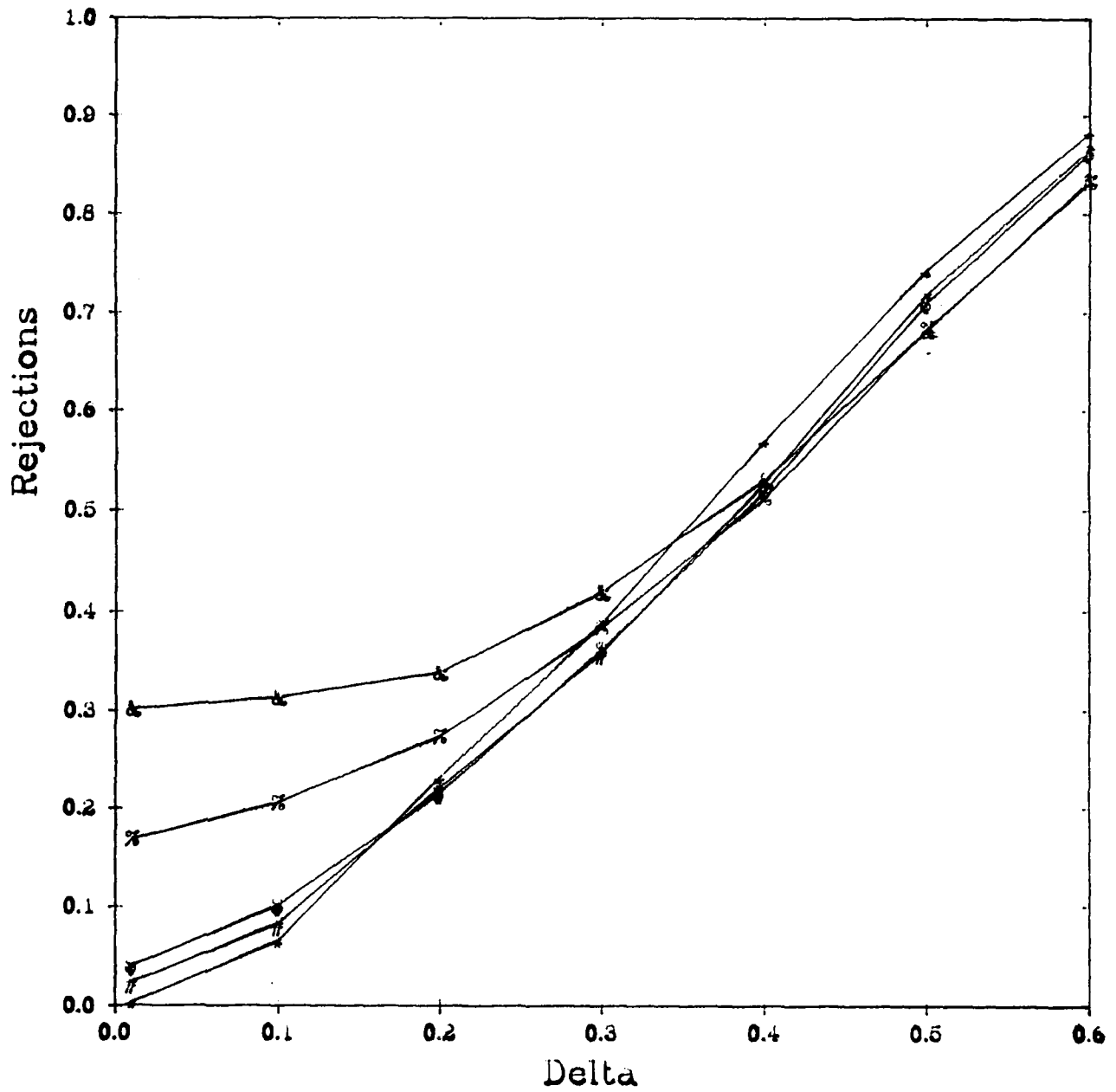
IAT=100 TO = 30-+ : 70-# : 100-\$: 200-% : 300-&



Graph 13

REJECTIONS OF TRANSACTIONS FROM CLASS #2

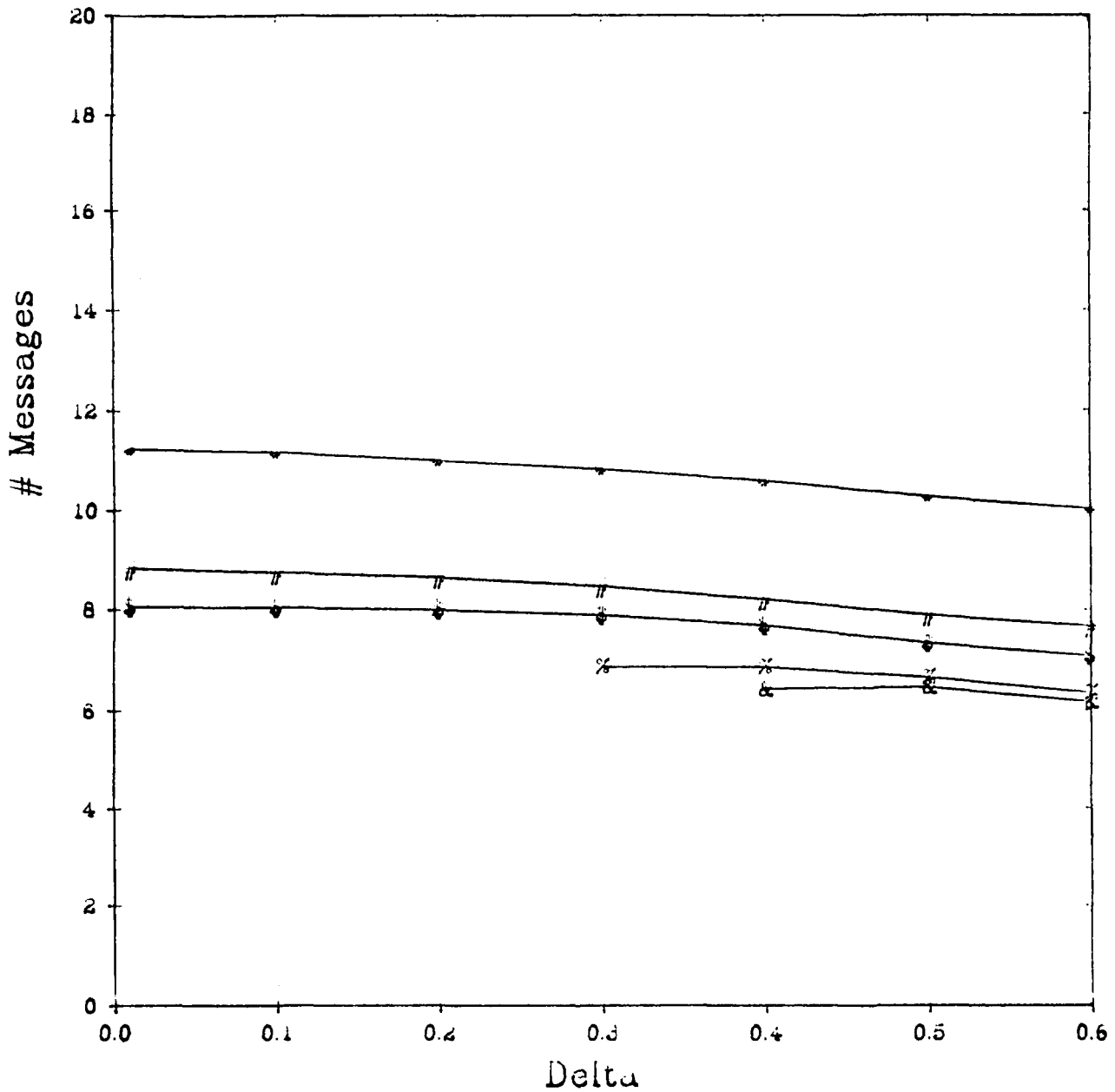
LAT=200 TO = 30-* : 70-# : 100-\$: 200-% : 300-&



Graph 14

AVG. NUM. OF MESSAGES SENT PER TRANSACTION

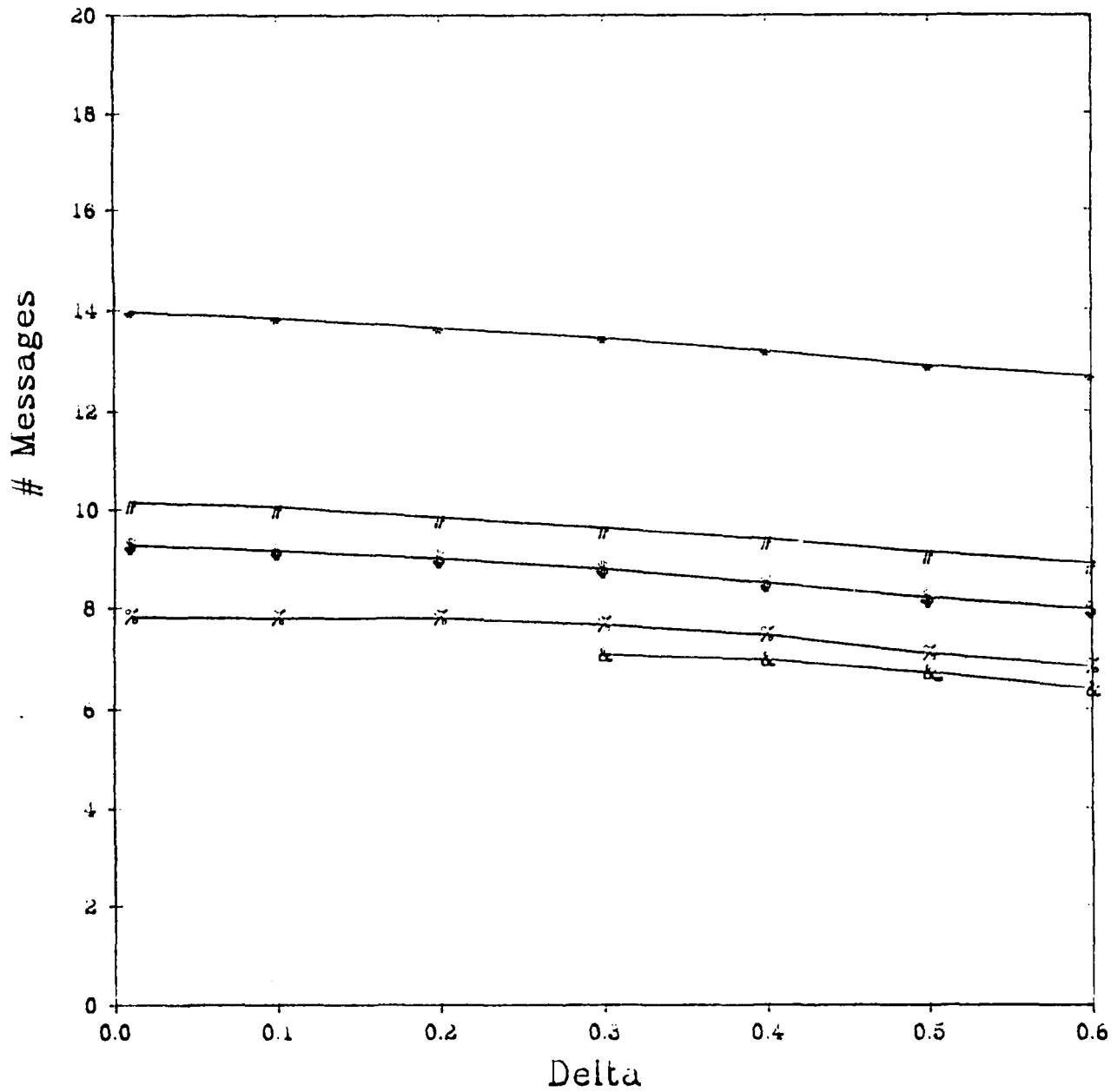
LAT= 60 TO = 30-+ : 70-# : 100-\$: 200-% : 300-&



Graph 15

AVG. NUM. OF MESSAGES SENT PER TRANSACTION

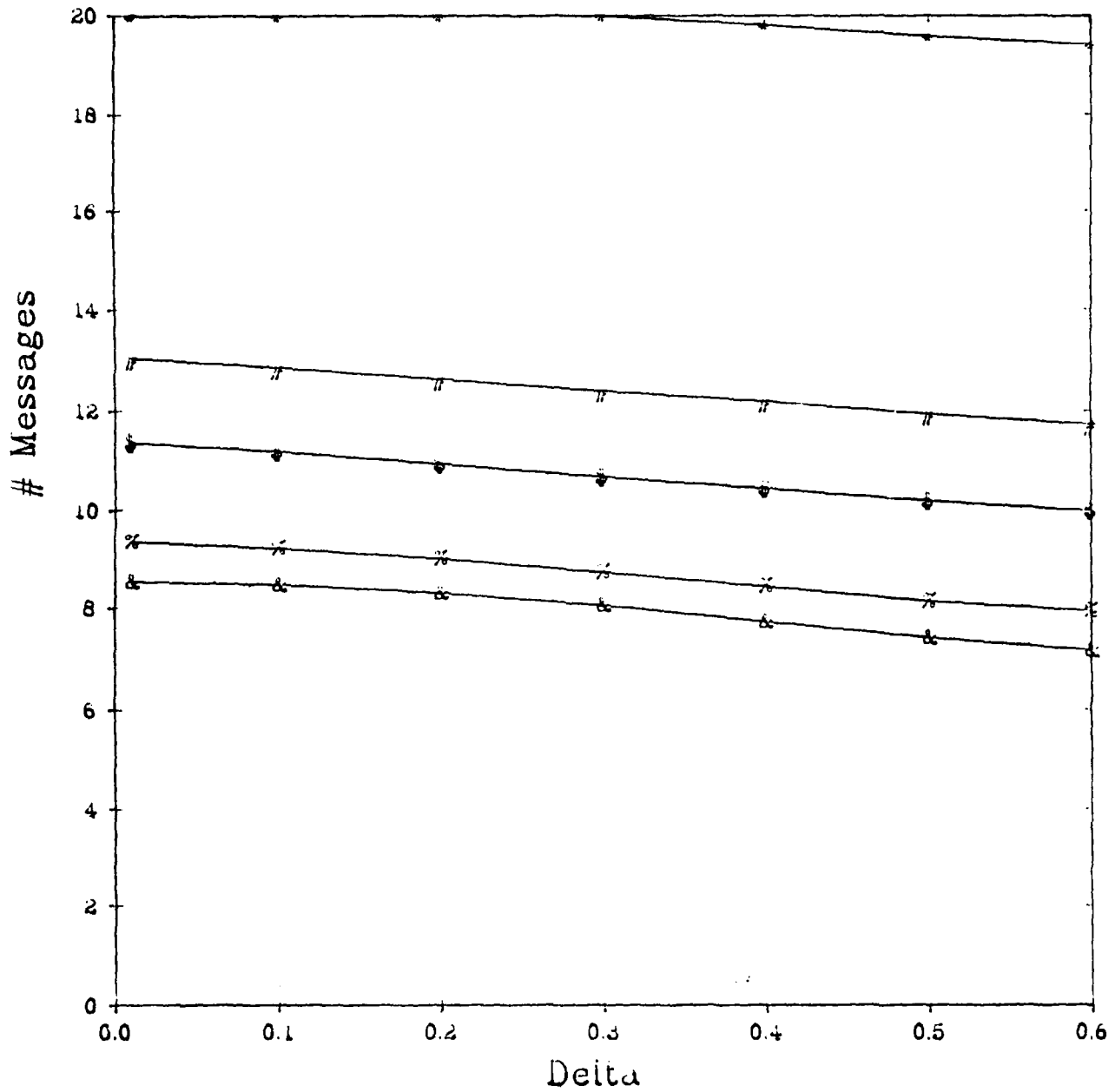
IAT=100 TO = 30-~ 70-// 100-\$ 200-% 300-&



Graph 16

AVG. NUM. OF MESSAGES SENT PER TRANSACTION

LAT=200 TO = 30-* : 70-# : 100-\$: 200-% : 300-&



Chapter VII

CONCLUSIONS AND RECOMMENDATIONS

The preliminary performance measurements presented in this report are a strong indication of the value, flexibility and power of full system simulation using a modular simulation tool. In effect, this report covers two interrelated areas of activity. The DISS package proved to be a very flexible tool and proved itself by the relative ease with which the simplified model was modified to the full fragmented data base model. The curtailment of the program did not permit the proper completion of the intended goals of the development effort as originally intended. It is suggested that an effort be made to prepare the necessary documentation of the two programming sections. The DISS package, as a generalized simulation aid, is intended for use for any distributed system simulation.

The simulation program of the SDD-1 algorithm, prepared to run with the DISS package, should be documented as well as this is but a research tool for further development of the algorithm. In addition, a competing algorithm, such as some version of the two phase locking scheme, should be implemented in the same simulation environment so that an objective comparison of the efficiency and costs of the different schemes may be made.

REFERENCES

- Bern78 Bernstein, P. A., Goodman, N., Rothnie, J. B., and Papadimitriou, C. A. "The concurrency control mechanism of SDD-1: A system for distributed databases (the fully redundant case)," IEEE Trans. Softw. Eng. SE-4,3 (May 1973), 154-169.
- Bern79a Bernstein, P. A., and Goodman, N. "Approaches to concurrency control in distributed database," in Proc. 1979 Natl. Computer Conf., AFIPS Press, Arlington, Va., June 1979.
- Bern79b Bernstein, P.A., Shipman, D. W., and Wong, W. S. "Formal Aspects of Serializability in Database Concurrency Control," IEEE Trans. Softw. Eng. SE-5,3, May 1979, 203-215.
- Bern80a Bernstein, P. A., and Shipman, D. W. "The correctness of concurrency control mechanisms in a system for distributed databases (SDD-1)," in ACM Trans. Database Syst., Vol. 5, No. 1, March 1980, 52-68.
- Bern80b Bernstein, P., Shipman, D.W., and Rothnie, J. B. "Concurrency control in a system for distributed databases (SDD-1)" in ACM Trans. on Database Syst., Vol. 5, No. 1, March 1980, 18-51.
- Bern81 Bernstein, P. A., and Goodman, N. "Concurrency control in distributed database systems," in IEEE computing Surveys, Vol. 13, No. 2, June, 1981, 185-221.
- McLe81 McLean, G. Jr., "Comments on SDD-1 concurrency control mechanisms," in ACM Trans. on Database Systems, Vol. 6, No. 2, June 1981, 347-350.
- Mena80 Menasce, D. A., Popek, G. J., and Muntz, R. R. "A locking protocol for resource coordination in distributed databases," in ACM Trans. on Database Sys., Vol. 5, No. 2, June 1980, 103-138.
- Rahi79 Rahimi, S. K., and Franta, W. R. "A posted update approach to concurrency control in distributed database systems," in proc. 1st Int. Conf. Distributed Computing Systems (IEEE), New York, Oct. 1979, pp. 632-641.
- Roth80 Rothnie, J. B., Jr., Bernstein, P. A., Fox, S., Goodman, N., Hammer, M., Landiers, T. A., Reeve, C., Shipman, D. W., and Wong, E. "Introduction to a System for Distributed Databases (SDD-1)," ACM Trans. on database systems, Vol. 5, No. 1, March 1980, 1-17.

- Russ73 Russell, E. D. Editor (1973). "SIMSCRIPT II.5 Programming
Language," C.A.C.I Inc.
- Thom79 Thomas, R. H. "A solution to the concurrency control problem
for multiple copy databases," in Proc. 1978 COMPCON
Conf.(IEEE), New York.

Appendix A

THE DISS METHOD OF DISTRIBUTED SYSTEM SIMULATION

Miron Livny and Myron Melman
Department of Applied Mathematics
The Weizmann Institute of Science
Rehovoth, Israel

A.1 ABSTRACT

The growing activity in the area of complex distributed systems has introduced the need for simulation aids that enhance the modeling time of these systems. A simulation modeling tool based upon the Process concept of SIMSCRIPT II.5 and addressing itself to Fully Distributed Processing systems is presented here. The package is based upon the principles of loosely coupled nodes displaying a cooperative autonomy in their internodal relationship. The two levels of modularity used offers flexibility and extensibility of the models. A modeling procedure and example are presented.

A.2 1. INTRODUCTION

Fully ² distributed processing systems are a growing phenomenon. Their size, complexity and cost are continuously on the increase. The advisability to construct such a system without having it undergo a complete simulation is reduced as the system complexity grows. The need for simulation aids that particularly address the needs of such systems is very current. This paper describes a software package that was developed for the express needs of simulating distributed processing systems. DISS, Distributed System Simulator, was designed to support and complement the modelers efforts to simulate multiple node, loosely coupled networks that can be depicted as directed graphs of arbitrary topology.

Modularity and extensibility are the major advantages of fully distributed systems. A performance study of such a system will undoubtedly include an analysis of the impact of topological changes and replacements of components on the performance of the system. Therefore it is desired that a simulation model of such a system will also be modular and extensible. An attempt was made in this package to permit the modeler full independence in the choice of the algorithmic description of the system, while using modular structures to build the model. DISS was written in the SIMSCRIPT II.5 programming language (1). Previous efforts to design simulation models of distributed systems have resulted in major projects and their successful, non-modular organization made modifications difficult and their use for near-like models impossible (2),(3). The latter model had network topology flexibility and the nodes represented pip-11

² This research was supported by the United States Air Force, Air Force Office of Scientific Research under Grant No. AFOSR 81/0147.

systems with parametric flexibility. But the attempt to replace the CPU meant to redesign the entire model. Other simulation programs and packages have been designed for the express purpose of simulating communications protocols (4),(5).

The underlying motivations and principles of the type of system organization with which the package is intended to be used are reviewed below, and the content and services supplied by the package routines are discussed. The services provide a cohesive set of functions that are common to such distributed systems displaying decentralized control (6). This package capability enables the modeler to concentrate on the use of a highly modular method of algorithmic descriptions which contain the linkages to the services supplied by the DISS package. The package considers the node as the building block of the model. The specific architecture suggested for the node is based upon an implementation of a unique 'WAIT UNTIL' type of statement used when alerting nodal members of a model. The organization of the 'WAIT UNTIL' strongly suggests a 'PROCESS' type of architecture for the node (7). This style is particularly convenient in handling loosely coupled nodes that can accommodate multiple resources and concurrent timing delays.

Section 2 of the paper describes the background experience that motivated the preparation of the package. The overall structure of DISS is presented in Section 3, and some of the critical design features of a model are discussed in section 4. The suggested architecture of the node process is presented in section 5. A description of a complete simulation study of a system is described in section 5 and the implemen-

tation of the modeling procedure on a sample model is presented in section 7. The experience of the authors and the direction for future development is discussed in the conclusions of section 9.

A.3 2. MOTIVATION FOR PACKAGE DESIGN

The organization of DISS has been strongly influenced, if not guided, by the modeling and simulation experience of the authors and by the simultaneous formulation by Enslow of the definitions of Distributed Processing Systems (6). The simulation studies mentioned above were of distributed processing systems, but their rigid, non-modular design made the models useful only for their single intended experiment. It became clear that these models had many common operating characteristics, yet, it was impossible to share sections of one model with sections of another model. The models were of distributed computers and of computer networks. As such they are depictable as directed graphs. These distributed system models, without a centralized controller, allowed complete autonomy to each node, such as to accept or reject tasks, and there were global directories so that the task in the node did not see the entire system.

The explicit features of such distributed systems have been aptly detailed by Enslow as Fully Distributed Processing Systems, (FDPS), distinguishing characteristics (3); the multiplicity of system resources, the interconnection of the nodes, the commonality of nodal controls, transparency of the system to the task and the autonomy of the node. Of these, the most distinguishing attribute was singled out as the "cooperative autonomy" of the node. It was realized that the models developed

were functionally abiding by these characteristics but the total lack of modularity made the models rigid and inflexible. A new approach was sought that would encompass the desired system functioning characteristics while at the same time permit a simplification of the modeling procedure and allow a greater degree of flexibility in the implementation. It is the belief of the authors that the DISS package presented in this paper meets these aspirations.

The coupling structure between nodes is critical in FDP Systems and is reflected in the design of all nodal elements. Distributed systems are characterized by the autonomy of their elements. The elements of the system are interconnected by a communications network through which they exchange information. The means by which the communication system transfers data, both physically and logically, should not violate the autonomy of the system elements. Therefore a fully distributed system has to be both physically and logically loosely coupled. The physical transfer of data, the physical coupling, has to be performed via a shared 'mail box' that is not an integral part of the system elements. In a distributed computer system the 'mail box' should be an on-line Input/Output device but not the primary memory of a computer. The autonomy of an element prohibits any external access to its internal storage components. It is the duty of the logic associated with the transfer mechanism, the logical coupling, to determine what has to be done with the data stored in the 'mail box'. It is the receiving element that decides, according to a well-defined two-party cooperative protocol, whether to accept the data or not. The development of DISS was guided by the idea that the 'loose physical coupling' of the distri-

buted system should be reflected in the structure of the model that is supported by DISS, and that the modeler should be provided with means that will enable him to model loosely coupled logic.

The basic and underlying assumptions in the development of DISS is that all such FDP Systems can be mapped into directed graphs. The modeler is left with the task of formulating his distributed system into a set of unique discrete nodes. At the outset of the experiment, when establishing the network in preparation for simulation the nodes are placed in their relative positions in the graph and are interconnected by the arcs. The mechanism used for transferring information along the arcs enable the node to maintain its physical and logical autonomy. The mapping conforms to the topology selected by the modeler.

A.4 3. THE STRUCTURE OF DISS

The package has been organized as a set of subroutines that are called by the model at various stages of the simulation experiment, to perform those operations that are common to all FDP Systems. The DISS routines are general in nature but they clearly address themselves to systems being modeled that display the above mentioned characteristics. The routines are grouped into activity areas that are shown in Fig. 1. Each node of the system model is capable of supporting any number of privately managed or shared resources and is connected by means of arcs to its neighbor nodes. These arcs carry the internodal communications for the system. The node of the network is designed to have its own control rules, and is determined by the system requirements. Each node of the system may be unique or all nodes may be identical. However, there may

be certain overall system control rules that are common to all nodes. System transparency and element autonomy are the powerful levers of control that the model imparts to the node. The degree of this control and the style of its implementation is left to the modeler. DISS gives the modeler the impression that he is using a language that is in both the functions and data structures it provides one level higher than SIMSCRIPT II.5. All these structures and functions, however, are obtained by the use of SIMSCRIPT II.5 statements.

SYSTEM ESTABLISHMENT
STATE VARIABLES
EXPERIMENT CONTROL
LOCAL STRUCTURE ALLOCATION
TIMING
NODE COUPLING
REPORT GENERATION
DEBUGGING AIDS
STATISTICAL ANALYSIS

Figure 1 Activity Areas of DISS

As a result of familiarization with DISS, the modeler should have an understanding of the characteristics of FDPS, those services provided by the package and therefore a picture of how and when these services are called for from within the model. This then leaves the modeler to concentrate on the needs of the particular system to be modeled and to design the unique instances of network nodes. Therefore the amount of code that the modeler need prepare and debug is reduced.

A.5 4. NODAL COUPLING

DISS regards the node of the model as a physical and logical autonomous element. In order to enable the exchange of data between two nodes DISS provides means for the physical coupling of the nodes that can be controlled by the logic of the nodes, the protocol. In the directed-graph representation of the model the arcs of the graph represent the physical coupling between two nodes. A directed arc that goes from one node, the source, to another, the target, represents the ability of the source to transfer information to the target. In a discrete event model the transfer of data from one node to another is associated with a change in the state of the source. The information transferred describes a change in state like 'start', 'end of message transfer', 'task arrival', 'buffer full', 'server not busy' etc.

DISS implements each arc as a group of variables, an entity, that can be considered as a 'mail box' into which the source writes and from which the target reads. The arc consists of a standard set of variables and the modeler can increase the number of variables according to the requirements of his model. DISS considers two types of information transfers, active transfer and passive transfer.

A.5.1 Active Transfer

When an active transfer of information takes place the data is stored in the 'mail box' and an attempt is made to ALERT the target node. The change in the state of the source is considered an External Event for the target. The target might be waiting for this event and therefore has to be alerted as the event takes place. The data stored in the 'mail box' describes the change in the state of the source. Each node is provided with means by which it can dynamically enable and disable alerts of selected External Events. If the event that caused the active transfer was masked by the target node the alert will be pended so that when the mask is removed the target node will be alerted. The masking mechanism thus ensures the logical autonomy of the nodes. One node can not force another node to do anything unless the node wants to cooperate. The active transfer of data is carried out by the ALERT routine that receives the description of the event via parameters. The data structures associated with the transfer are established by the DISS routines at network establishment.

A.5.2 Passive Transfer

A node may be interested in making certain state variables visible to another node while changes in these variables will not cause an External Event. In such cases a change in one of the state variables that describe these aspects will cause a passive transfer of data. The transferred data is placed in the 'mail box' by the source node. The variables used by the passive transfer are called the INTER NODAL STATE VARIABLES of the arc. Each incoming or outgoing arc is considered as a

port of the node. The data is sent and received according to port numbers. Each input port can be assigned a priority so that the data transfer activity can utilize priority schemes.

The nodal coupling of DISS does not rely on any structural dependency between the source and target nodes. The source node can transfer data without knowing anything about the structure of the target. Thus any two nodes that follow the same protocol can exchange information along an arc.

A.6 5. THE PROCESS

DISS considers a SIMSCRIPT PROCESS as the unit of nodal activity for the model. There may be a single Process type for all nodes or there may be a unique Process type for every node. In all combinations of Processes, they must all contain common structural and interfacing statements that link them to DISS. To control the experiment the modeler need prepare a special Process, called Executive Manager, (EM), that manages the sequencing of the complete experiment. Fig. 2 is a block diagram of a typical Process organized to operate with DISS. The structure is capable of maintaining the distinguishing characteristics of a Fully Distributed Processing System. The focal point of the organization is the loop structure that is controlled by the "WAIT FOR ALERT" function. This represents a generalized "WAIT UNTIL" implementation and is the key to the Process operation (9).

The "WAIT FOR ALERT" function of DISS enables the Process to enter a passive state until one of a number of events will take place. The

"WAIT FOR ALERT" mechanism is equivalent to a generalized "WAIT UNTIL" with a condition that is a logical concatenation of the External Events of the node and of the internally scheduled nodal delays. The mechanism transfers the imperative scheduling of SIMSCRIPT into an interrogative scheduling mechanism. The interrogative scheduling of DISS is the foundation of the autonomy of the node.

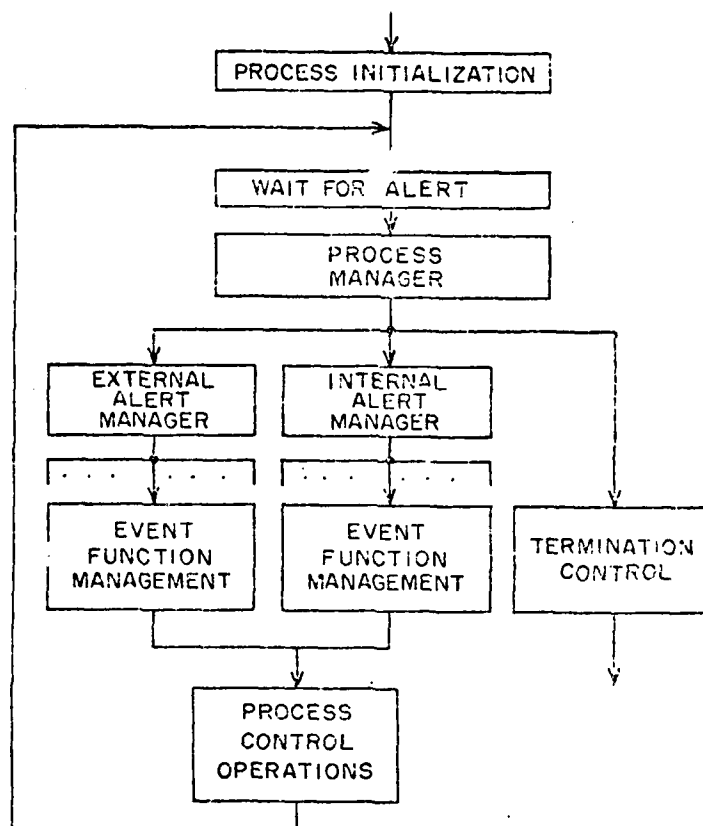


Figure 2 Typical Process Structure

The Process is activated by DISS and is permitted to execute its initialization period. During this operation the Process uses DISS to establish the local structures that will be used during the experiment and to obtain from the modeler any personalizing characteristics required by the node. Simulation may not begin until all Processes have been initialized so it is convenient to have the Process suspend at the end of this interval. The "WAIT FOR ALERT" function serves this purpose at this time.

As mentioned in the last section, the coupling mechanism has the capability of alerting a Process when implementing an External Event. The Process, as shown, is structured to receive three types of alerts; External, Internal and termination. The External alerts are invoked by neighbor nodes that communicate over the interconnecting arcs and represent the termination of message transmission to this node. Internal alerts are terminations of time delays that have been scheduled by this Process while the termination alert is scheduled for all Processes by the EM at the instance of the end of simulation time.

At the entry to the 'Wait For Alert' state from either the initialization path or from the iterative loop path, the Process logic invokes a check for the presence of a pending alert. DISS provides the means for automatically testing for the presence of such an alert, in which case the Process continues to execute, or for the absence of such an alert, in which case the Process suspends. Once suspended, the Process waits for the next event to occur. In case the Process has been suspended, DISS provides the automatic means by which an event alert may resume the Process.

When designing the Process the modeler has several options for the processing of event alerts. The modeler can use either a single mask flag for global Process event enabling or he may select the use of a more sophisticated mask vector that would relate each mask element to a particular event handled by the Process. If the global mask is selected, DISS provides all support required. If the latter method is chosen then the modeler must provide a routine, called EV.SELECT, in which the coded event alert identification is matched against the related enabling mask vector element. The sequence of selecting the pending active event alerts for acceptance is determined by a scheduling or priority algorithm suitable to the needs of the Process. If this invoked procedure occurred when the Process was suspended and an active alert is accepted, the Process is resumed.

From Fig. 2 it is seen that once resumed the Process selects any one of a number of execution paths that algorithmically describe the needs of the node as a reaction to the particular motivating Internal or External Event. The functions are modular within the Process and are simple to remove or insert. The internal design of the function is also modularly structured and is simple to design and manipulate. In general, the functions are used to generate additional activities; monitor system, node and function status; and make the required measurements pertinent to the motivating event. The modular function is then clearly linked to the event that caused the iteration through the Process. In keeping with the SIMSCRIPT language, each Process and function may have its own resources. The loop structure and the multiple parallel path concept enable the modeler to introduce simultaneous, concurrent time

delays within the same Process. This can represent concurrent message transmission over different arcs, or concurrent utilization of multiple resources within the Process.

Report generation at the end of the simulation run, per Process, is initiated by a termination event scheduled by the Executive manager. The modeler is free to organize and print the performance data as best suites the model. The termination function should also handle the release and resetting of those SIMSCRIPT structures that would be required to permit the outer looping of additional system runs. The Process may then be self destroyed.

A.7 6. SIMULATION STUDY WITH DISS

The interdependent modeling concepts to represent FDP Systems with the use of the DISS package have been described in the preceding sections. The various stages of formulating a simulation study of a specified system can be visualized by a sequence of clearly defined operations. Fig. 3 shows the conceptual, planning and implementation phases used to achieve such a study.

The system under consideration either exists or is somehow specified. In either case a detailed analysis of the system is to be made so that there is a clear understanding of the operations executed by the system, of the characteristics of the system resources, and of the nature of the workload. The scope of the experimental frame (10) should be clear from the outset of the study, why simulate, what performance information is expected from the study.

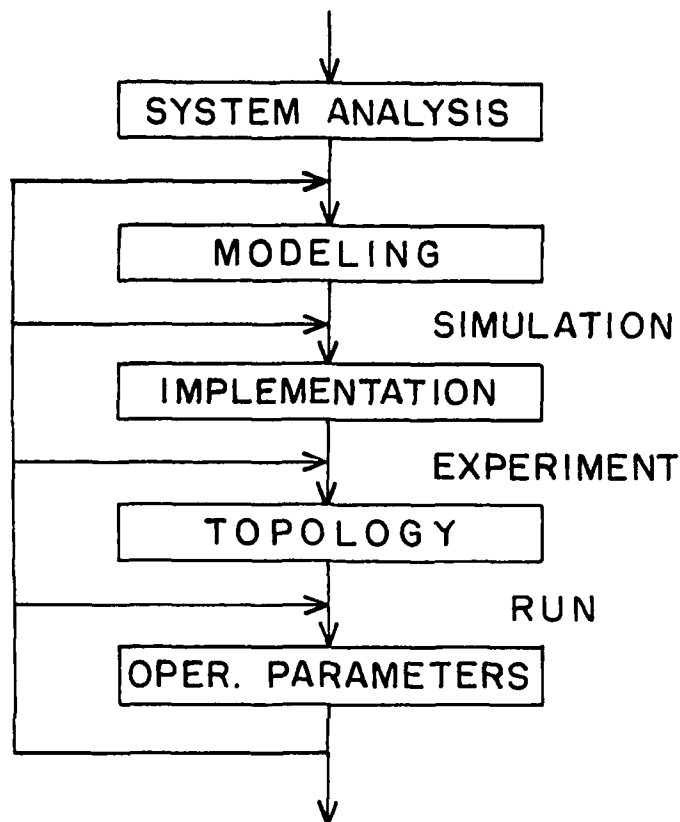


Figure 3 Phases of Simulation Study

Once analyzed, the system is ready to be modeled. It is at this point that the choice of simulation methodology is to be made. For the instance of this paper, the modeler has SIMSCRIPT II.5 and is familiar with DISS. The modeler should have a clear picture as to the alternatives available for selecting the modeling units to represent the building block elements of the real system. The leading decision to be made is how many unique Process types are required to model the system. There may be several answers, but one of them will give the optimal overall simulating conditions. The number of unique node types can have a

direct effect upon the topology selected, so that the resulting topology of the model may bear no resemblance to the distributed network used in the physical system (11). Upon determining the number of unique nodes, the modeler is left with the task of using the system analysis information to map the system algorithms into modular functions to be designed into the node. These are to be grouped into externally and internally invoked events. The internodal state variables for each arc may then be defined.

The implementation phase begins with the coding of the functions and the grouping of these modular sections into the node Process shown in Fig. 2. During this stage, the modeler places the various interfacing calls to invoke DISS services. When the node Processes are complete the modeler will organize the Executive Manager Process. By this time the modeler should have a clear picture of the sequencing of the simulation experiment. These global controls may effect the nodal functions in which case the modifications are made to the nodes. Note that in this phase of implementation both function and node Process are modular. It is this feature that ensures the extensibility of the model.

The model, written in SIMSCRIPT II.5, is now ready to be run. It remains for the modeler to prepare the topology of the model in the required format and to organize the personalizing parameters for each of the node Processes, those global parameters oriented to the modeled system and those parameters required by DISS. When prepared, the program is ready to be run. The modeler will manage the experiment as best suits the study.

As can be seen from Fig. 3, the preparations of the parameters infers a simulation run. Depending upon the demands of the experimental frame it may be possible to execute statistical analysis after a single run or it may be necessary to make parametric changes and repeat the run several times before executing the statistical analysis. After the analysis it is usually desirable to compare the results with additional topological configurations. In that case the loop into the topology is used, the change in topology is made and the inner loop is repeated. The topology loop may be repeated according to the needs of the experimental frame.

It may be required to change the experimental frame which will cause a change to the model structure. In that case the loop is extended to the next level and the modular implementation features of the model will permit a fast program modification. The model may then be reactivated as though from the start. The outer loop calls for a change to the model concept with means a basic change in the system analysis.

A.8 7. EXAMPLE

Assume that a simulation study of the following system has to be performed and that the model of the system will be implemented using DISS. The modeling phase of the study is presented step by step below. The system to be modeled is a distributed processing system that consists of a number of HOSTS that are interconnected by a message switching store-and-forward communication system as shown in Fig. 4. The network is made up of communication processors, CP, that are connected by full-duplex communication lines. Each CP has a finite buffer space into which

the messages are stored. Therefore the communication protocol performs a 'space reservation' step before a message is transmitted. Each HOST receives an independent stream of tasks. Every task is assigned an execution site at which it will be served. This assignment is performed by the resource allocation algorithm of the distributed system.

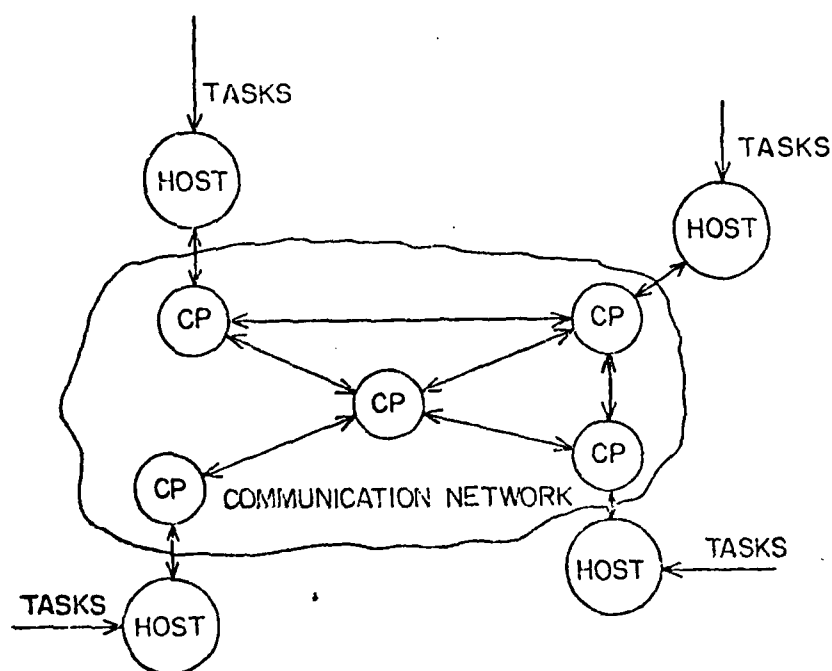


Figure 4 The Distributed System

Given the definition of the system the modeling phase is executed by the ensuing steps:

A.8.1 Node Definition

The elements of the above system may be grouped into nodes in a number of ways three of which will be listed here:

1. Each element of the system defines a node. The model will include two types of nodes.
2. The HOST and the CP are grouped into one node so that the model has only one type of node. An input parameter will determine whether the node is a HOST, a CP or both.
3. A HOST defines one type of node whereas all the communication processors of the network are grouped into a second node type. This second node will represent the entire network. In this case the topology of the network will be represented internally by this node.

The selection of a mapping scheme depends strongly on the experimental frame of the study. Each of the above schemes can be considered as the best under the requirements of a different study. One scheme may be more modular whereas another one may have a more efficient implementation. A detailed analysis of the above schemes is beyond the scope of this paper. For the purpose of this example it will be assumed that the first scheme has been selected.

A.8.2 Internodal State Variables

After the elements are grouped into nodes the External Events of each node and the internodal state variables have to be defined. Each arc of the model will include the following state variables:

1. BUFFER.FULL indicates the state of the message buffer of a CP.
2. WAIT will be set whenever the node wants to transfer a message along the arc and the buffer of the target node is full.

These two variables are used for the passive transfer of state information between the nodes.

A.8.3 External Events

The External Events of the CP nodes will be the following:

1. START OF MESSAGE ARRIVAL. The source node has started the transmission of a message.
2. END OF MESSAGE ARRIVAL. The source node has terminated the transfer of a message.
3. BUFFER AVAILABLE. This event takes place whenever the state of the buffer changes from full to available and the target node is in a wait state.

The set of External Events of the HOST node will include only two out of the above three events. The START OF MESSAGE ARRIVAL Event is not considered as an External Event by the host. The reservation step per-

formed by the CP as a result of this event are not part of the host functions.

Note that this structure of the model gives the CP full autonomy in allocating available buffer space. By means of the BUFFER.FULL variable and the BUFFER AVAILABLE event it can select which waiting CP will be given a buffer space that has become available.

A.8.4 Internal Events

Before each node can be implemented as a process the Internal Events of each node have to be defined. The HOST node includes the following Internal Events:

1. END OF MESSAGE TRANSMISSION. This event represents the delay associated with the transfer of a message.
2. END OF TASK EXECUTION. The end of the execution period of a task is represented by this event.
3. TASK ARRIVAL. The arrival procedure of the tasks is modeled by this event. The arrival of one task causes the scheduling of the next arrival.

The CP has only the END OF MESSAGE TRANSMISSION event. The above set of Internal Events enable the HOST to control a number of service-resources simultaneously. The CP can simultaneously transfer a number of messages along different arcs.

In this example the outline of a model of the above distributed system was represented. It is clear that the model defined above is not the only way such a system can be modeled and implemented by DISS.

A.9 8. CONCLUSIONS

In this paper, the authors have shown that by using an integrated design concept the modeling effort of Fully Distributed Processing systems may be minimized. The loop structure of the node, the "WAIT FOR ALERT" implementation, the loose coupling of the nodes and the timing mechanism of DISS form that integrated concept.

The DISS package has been in operation for some time now. The experience of the authors indicates that its use has considerably enhanced the challenge of modeling such systems. Two major simulation studies were undertaken with the use of DISS: The first was a study of the ETHERNET. Besides preparing an Executive Manager, the ETHERNET system was conveniently modeled into two Processes, the 'Physical Layer' and the 'Data Link Layer'. The latter Process can then be used for as many nodes as desired. The topology ended up as a star network with the 'Physical Layer' as the focal point. The complete basic model was then replaced by a lumped model (10) of the system to compare the system performance under these two modeling conditions. This experiment was the basis for testing three load balancing algorithms (11).

The second system is a model of the SDD-1 Concurrency Control Algorithm representing a fully distributed multiple copy data base system. Again, besides the EM only two Processes, the 'Transaction Manager' and

the 'Data Manager' were written. These models have been run for different numbers of nodes and in several network topologies.

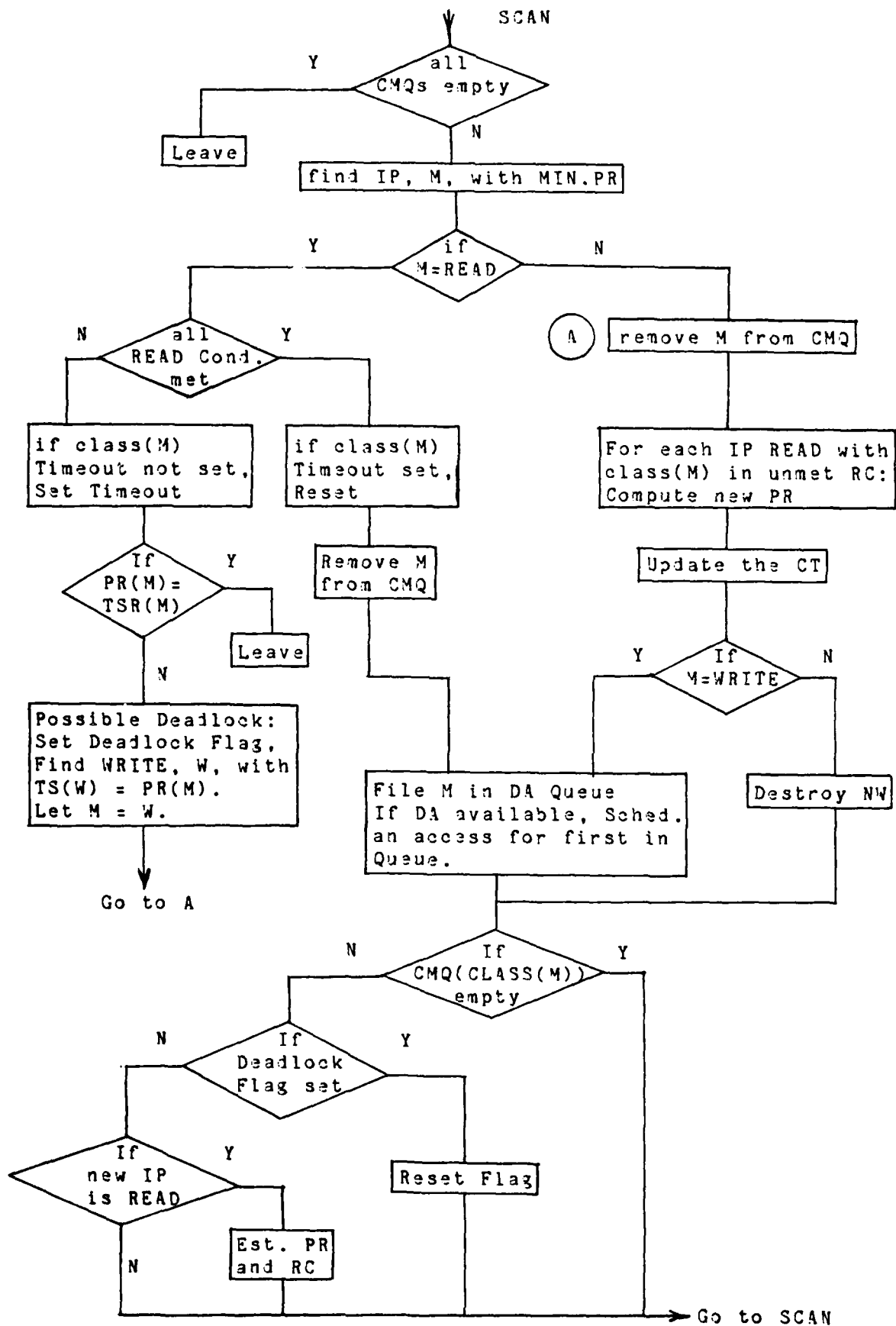
Continued work with DISS will broaden the library of routines available to the modeler and will include sets of modular nodes and modular functions. Additional simulation aids in the field of report generation and statistical analysis will enrich the package and offer the modeler a wider range of options.

REFERENCES

- RUSSELL, E.D. Editor SIMSCRIPT II.5 Programming Language. C.A.C.I. Inc. 1973
- BARAK, A.B. and MELMAN, M. "Minimal Configuration of a Flexible Expandable Distributed Processing System." Information Technology (Proc. Jerusalem Conference on Information Technology, 3rd, Moneta, J. Editor, North-Holland Publishing Co. p 27-31. 1978
- FENIG, B. and MELMAN, M. "Simulating a Distributed Computer System Network With a Generic Modeling Tool." Proc. IMAC European Simulation Meeting University of Patras, Patras, Greece, Oct. 2-4, 1979, Tzafestas, S.Y. Editor, North-Holland Publishing Co. 1980
- SCHNEIDER, G.M. "A Modeling Package for Simulation of Computer Networks." Simulation December 1978, p 182-192. 1978
- SHOEMAKER, S. Editor Computer Networks and Simulation. North-Holland Publishing Co. 1978
- ENSLOW, P.H. "What is a 'Distributed' Data Processing System?" Computer January 1978, p 13-21. 1978
- RUSSELL, E.D., Simulating with Processes and Resources. C.A.C.I. Inc. 1975
- ENSLOW, P.H. and SAPONAS, T.G. "Distributed and Decentralized Control in Fully Distributed Processing Systems." Final Technical Report GIT-ICS-81/02, Georgia Institute of Technology, School of Information and Computer Science, Atlanta, Georgia 30332, 1981
- FRANTA, W., A Process View of Simulation North-Holland, New York, 1977
- ZEIGLER, B.P. Theory of Modelling and Simulation. John Wiley & Sons, New York 1976
- LIVNY, M. and MELMAN, M. "Load Balancing in Homogeneous Broadcast Distributed Systems." Proc. ACM Computer Network Performance Symposium April 13-14, College Park, Maryland, 1982

Appendix B

SCAN - BLOCK DIAGRAM



Appendix C
SIMULATION RESULTS

CONCURRENCY CONTROL ALGORITHM

R - W SYNCHRONIZATION CONSERVATIVE T/C
 W - W SYNCHRONIZATION T W R
 =====

NETWORK TOPOLOGY INPUT
 =====

NETWORK CHARACTERIZATION - NEIGHBOR LIST
 =====

1 (5, .50) (6, 9.50)	2 (5, .50) (6, 9.50)
3 (5, 9.50) (6, .50)	4 (5, 9.50) (6, .50)
5 (1, .50) (2, .50) (3, 9.50) (4, 9.50)	6 (1, 9.50) (2, .50)

ROUTING MATRIX
 =====

NODE.ID

	1	2	3	4	5	6
1	1	5	5	5	5	6
2	5	2	5	5	5	6
3	6	6	3	6	5	6
4	6	6	6	4	5	6
5	1	2	3	4	5	1
6	1	2	3	4	3	6

D I S S

18.36.28 02- 5-82 PAGE- 1

ATIVE T/C

=====

2 (5, .50) (6,9.50)
4 (5,9.50) (6, .50)
6 (1,9.50) (2,9.50) (3, .50) (4, .50)

RUN NO. 1 : SIMPLE MODEL , TRACE , LOGS
=====

INPUT FOR THE RUN
=====

GLOBAL PARAMETERS
=====

TIME UNITS	MILLISECONDS
NUMBER OF CLASSES IN THE SYSTEM	4
MAX NUM OF FRAGMENTS PER SFT	0
AVERAGE LENGTH OF A CONTROL MESSAGE	0.50 (UNITS OF TIME)
BATCH SIZE	25
CONFIDENCE LEVEL	0.90

FLAGS
=====

TRACE FLAG 15
TRACE DRIVEN FLAG 0
LOG FLAG 1

PROTOCOL NUMBERS
=====

CLASS	1	2	3	4
PROTOCOL	1	2	3	1

CONFLICT CLASS MATRIX
=====

	1	2	3	4
1	0	0	0	0
2	1	0	1	0
3	0	0	0	1
4	0	0	0	0

NODE CHARACTERIZATION
=====

NODE ID	PROCS TYPE	CLASS	TRAN INT ARR (EXPD)	S E D	AV PRCG. LENGTH (EXPD)	S E D	DATA ACCES TIME (EXPC)	S E D	TIME CUT (FIX)	DLI PCR FLG	READ TIME STAMP DELTA
1	TM	1	100.0	4	2.0	5			100.0	1	.2000
2	TM	2	100.0	6	2.0	7			100.0	1	.2000
3	TM	3	100.0	8	2.0	9			100.0		
4	TM	4	100.0	10	2.0	11			100.0	1	.2000
5	DM						3.0	2	100.0		
6	DM						3.0	3	100.0		

5

15.54.45 02- 5-82 PAGE- 2

TRACE , LCGS

TIME)

TIME CUT (FIX)	DLI PCR FLG	READ TIME STAMP DELTA
100.0	1	.2000
100.0	1	.2000
100.0		
100.0	1	.2000
100.0		
100.0		

SYSTEM PERFORMANCE RESULTS

SIMULATION TIME = 350.0 UNITS

TM NODE STATISTICS

NOD ID	NUM TRAN COMP	TRANS TM C. TIME	AVG # WAIT. TRANS	% CF REJEC TRANS	TM UTIL.	AVG RESPC TIME	RESPC TIME STD	BA AV ST
1	4	2.616	.030	0.	.364	34.444	38.174	
2	3	23.483	.201	0.	.714	106.820	18.221	
3	1	.000	0.	0.	.362	118.154	0.	
4	3	22.428	.192	0.	.317	59.404	36.381	

DM NODE STATISTICS

NOD ID	READ CMQ TIME	WRITE CMQ TIME
5	42.386	11.353
6	42.015	.000

GLOBAL STATISTICS

NUMBER OF TRANSACTIONS ARRIVED IN SYSTEM

NUMBER OF TRANSACTIONS PROCESSED (AFTER WRITES)

NUMBER OF TRANSACTIONS REJECTED

NUMBER OF MESSAGES SENT PER TRANSACTION , AVG (INC. NW)

AVG. SYSTEM TIME PER TRANSACTION

AVG. TRANSMISSION TIME PER TRANSACTION (NOT INC. NW)

AVG. RESPONSE TIME PER TRANSACTION

9
93
17
68

AVG RESPC TIME	RESPC TIME STD	BATCH AVG STD	RELAT FREQ IN %	CORR	NUM OF BATCHES
34.444	38.174				
106.820	18.221				
118.154	0.				
59.404	36.381				

12

11

C

. NW)

9.2500

93.7740 UNITS

M)

17.9439 UNITS

68.6003 UNITS

THE RUN NEEDED

1.85 CPU SECONDS

MS28TC01 FILE - 3

D I S S

TIME.V NCDE TR.NC I

EVENT, NAME = NUM1, TII

13.6	2	1	TANT,CLAS=	2					
13.6	2	1	TSRD,DEST=	5	1102				
14.6	5	1	CARC,CLAS=	2	1102				
14.6	5	1	DRPR,PRIC=	1102	1102				
14.6	5	1	DMFR,MSTP=	3	1102				
14.6	5	1						DWIC,CLAS=	
49.4	2	2	TANT,CLAS=	2					
78.2	4	3	TANT,CLAS=	4					
78.2	4	3	TSRD,DEST=	6	6304				
79.2	6	3	CARC,CLAS=	4	6304				
79.2	6	3	DRPR,PRIC=	6304	6304				
79.2	6	3	DMFR,MSTP=	3	6304				
79.2	6	3	DQCA,MSTP=	3	6304				
81.6	6	3	DECA,MSTP=	3	6304				
82.6	4	3	TERC,	=	0	6304			
82.6	4	3	TSPG,DEST=	6					
83.2	6	3	DEPG,SRCE=	4					
84.2	4	3	TEEX,SRCE=	6					
84.8	3	4	TANT,CLAS=	3					
84.8	3	4	TSRD,DEST=	6	8403				
85.2	6	3						DELD,SRCE=	4
85.8	6	4	CARC,CLAS=	3	8403				
85.8	6	4	DRPR,PRIC=	8403	8403				
85.8	6	4	DMFR,MSTP=	3	8403				
85.8	6	4						DWIC,CLAS=	
86.2	4	3						TEUP,SRCE=	6
86.2	4	3						TETR,	=
87.2	6	3						CAWR,CLAS=	4
87.2	6	3	DMFR,MSTP=	2	7804				
87.2	6	3	DQCA,MSTP=	2	7804				
87.2	6	4	DMFR,MSTP=	3	8403				
89.7	6	3	DEDA,MSTP=	2	7804				
96.2	5	3						CAWR,CLAS=	4
96.2	5	1	DMFR,MSTP=	3	1102				

TANT	ARRIVAL NEW TRANSACTION	DQCA	ENG TC DATA ACCESS QUEUE	DEPG	END CF
TSRD	SENDS READ MESSAGE	DECA	END CF DATA ACCESS	TEES	END CF
CARC	ARRIVAL OF READ MESSAGE	DMFR	END CF MSG TRANSMISSION	DELD	END CF
DMFR	SEND REJECT	TERD	END CF READ PHASE	TEUP	END CF
DMFR	MINIMUM PRIORITY	TSPG	SEND EXEC PROGRAM		

S S

15.54.45 04- 5-84 PAGE- 1

NT, NAME = NUML, TIMESTAMP

DWTC,CLAS= 2 1102

304

6304

4

E= 6

SRCE= 4

DWTC,CLAS= 3 8403

UP,SRCE= 6 7804

TEIR, = 0 7804

CAWR,CLAS= 4 7804

7804

CAWR,CLAS= 4 7804

UE	DEPG	END CF EXEC PROGRAM	TETR	END CF TRANSACTION
N	TEES	END CF EXECUTION PHASE	CAWR	ARRIVAL CF WRITE MESSAGE
	DELD	END CF UPDATE DISTRIB.	TARJ	ARRIVAL CF REJECT AS TM
	TEUP	END CF UPDATE	DRPR	READ PRIORITY

M52BTC01 FILE - 3

D I S S

TIME.V NCDE TR.NC I

EVENT, NAME = ALM1, TIME

TIME.V	NCDE	TR.NC	I	EVENT, NAME	TIME
100.0	1	0		TSNw,	
100.0	2	0		TSNw,	
100.0	3	0		TSNw,	
101.0	5	0		DAN	
101.0	5	1		DMPR,MSTP=	3 1102
101.0	5	0		DAN	
101.0	6	0		DAN	
110.0	6	0		DAN	
110.0	6	4		DMPR,MSTP=	3 8403
110.0	6	0		DAN	
110.0	6	0		DMPR,MSTP=	1 1302
110.0	6	4		DMPR,MSTP=	3 8403
110.0	5	0		DAN	
110.0	5	1		DMPR,MSTP=	3 1102
110.0	5	1		DQCA,MSTP=	3 1102
110.0	5	0		DMPR,MSTP=	1 1302
110.0	5	3		DMPR,MSTP=	2 7804
110.0	5	3		DQCA,MSTP=	2 7804
110.0	5	0		DMPR,MSTP=	1 8403
110.0	5	0		DMPR,MSTP=	1 10001
115.1	5	1		DECA,MSTP=	3 1102
116.1	2	1		TERC,	= 0 1102
116.1	2	1		TSPG,DEST=	5
116.9	5	1		DEPG,SRCE=	2
117.9	2	1		TEEX,SRCE=	5
118.9	5	1		DELD,SRCE=	2
119.5	4	5		TANT,CLAS=	4
119.5	4	5		TSRD,DEST=	6 10904
119.9	2	1		TELP,SRCE=	5 1302
119.9	2	1		TETR,	= 0
119.9	2	2		TSRD,DEST=	5 4402
120.1	5	3		DECA,MSTP=	2 7804
120.5	6	5		CARD,CLAS=	4 10904
120.5	6	5		DRPR,PRIC=	10904 10904
TANT	ARRIVAL NEW TRANSACTION			DQCA	END TC DATA ACCESS QUEUE
TSRC	SENDS READ MESSAGE			DECA	END CF DATA ACCESS
DARD	ARRIVAL OF READ MESSAGE			CEMT	END CF MSG TRANSMISSION
DSRJ	SEND REJECT			TERD	END CF READ PHASE
DMPR	MINIMUM PRIORITY			TSPG	SEND EXEC PROGRAM

S S

15.54.45 04- 5-82 PAGE- 2

NT, NAME = NM1, TIMESTAMP

TSNw, = 0 10001

TSNw, = 0 1302

TSNw, = 0 8403

DANw,CLAS= 1 10001

DANw,CLAS= 2 1302

DANw,CLAS= 3 8403

DANw,CLAS= 1 10001

DANw,CLAS= 2 1302

DANw,CLAS= 3 8403

102

1102

5

2

E= 5

SRCE= 2

UP,SRCE= 5 1302

TEIR, = 0 1302

804

E	DEPG	END CF EXEC PROGRAM	TEIR	END CF TRANSACTION
	TEES	END CF EXECUTICA PHASE	CAWR	ARRIVAL CF WRITE MESSAGE
	CEUD	END CF UPDATE DISTRIB.	TARJ	ARRIVAL CF REJECT AS TP
	TELP	END CF UPDATE	DRPR	READ PRICRITY

MS4BT001 FILE - 3

D I S S

TIME.V NCDE TR.AC I

EVENT, NAME = NCM1, T

120.5	6	4	DNFR,MSTP=	3	8403			
120.9	5	1				CAWR,CLAS=	2	
120.9	5	1	DNFR,MSTP=	2	1302			
120.9	5	1	DQCA,MSTP=	2	1302			
121.3	5	1	DECA,MSTP=	2	1302			
121.9	5	2	CARD,CLAS=	2	4202			
121.9	5	4	DRPR,PRIC=	4202	4202			
121.9	5	2	DNFR,MSTP=	3	4202			
121.9	5	2	DQCA,MSTP=	3	4202			
129.7	5	2	DECA,MSTP=	3	4202			
129.9	6	1				CAWR,CLAS=	2	
129.9	6	1	DNFR,MSTP=	2	1302			
129.9	6	1	DQCA,MSTP=	2	1302			
129.9	6	4	DNFR,MSTP=	3	8403			
130.7	2	2	TERC,	=	0	4202		
130.7	2	2	TSPG,DEST=	5				
131.1	6	1	DECA,MSTP=	2	1302			
131.2	5	2	DEPG,SRCE=	2				
132.2	2	2	TEEX,SRCE=	5				
133.2	5	2	DEUD,SRCE=	2				
134.2	2	2	TEUP,SRCE=	5	4			
134.2	2	2	TETR,	=	0			
135.2	5	2				CAWR,CLAS=	2	
135.2	5	2	DNFR,MSTP=	2	4502			
135.2	5	2	DQCA,MSTP=	2	4502			
138.9	5	2	DECA,MSTP=	2	4502			
139.8	4	6	TANT,CLAS=	4				
144.2	6	2				CAWR,CLAS=	2	
144.2	6	2	DNFR,MSTP=	2	4502			
144.2	6	2	DQCA,MSTP=	2	4502			
144.2	6	4	DNFR,MSTP=	3	8403			
146.2	6	2	DECA,MSTP=	2	4502			
181.3	2	7	TANT,CLAS=	2				
181.3	2	7	TSRC,DEST=	5	15402			

TANT	ARRIVAL NEW TRANSACTION	DQDA	ENQ IC DATA ACCESS QUEUE	DEPG	END CF
TSRD	SENDS READ MESSAGE	DECA	END CF DATA ACCESS	TEES	END CF
DARD	ARRIVAL OF READ MESSAGE	DEMT	END CF MSG TRANSMISSION	DEUD	END CF
DSRJ	SEND REJECT	TERD	END CF READ PHASE	TELP	END CF
DPFR	MINIMUM PRIORITY	TSPG	SEND EXEC PROGRAM		

I S S

15.54.45 02- 5-82 PAGE- 3

VENT, NAME = NUM1, TIMESTAMP

CAWR,CLAS= 2 1302

02

1302

02

4202

CAWR,CLAS= 2 1302

02

4202

5

1302

2

SRCE= 5

D,SRCE= 4

TEUP,SRCE= 5 4902

TEIR, = 0 4902

CAWR,CLAS= 2 4902

02

4902

CAWR,CLAS= 4 4902

02

4902

QUEUE	DEPG	END CF EXEC PROGRAM	TEIR	END CF TRANSACTION
SIGN	TEES	END CF EXECUTION PHASE	CAWR	ARRIVAL CF WRITE MESSAGE
	CEUC	END CF UPDATE DISTRIB.	TARJ	ARRIVAL CF REJECT AS TP
	TELP	END CF UPDATE	CRPR	READ PRICITY

MS2BTC01 FILE - 4

C I S S

TRANSACTION COMPLETION TABLE

TRANS. NO.	CLASS	ARR. TIME	IS	ISP	TM C.T	BCPR R.T	NTR.R CMC	LVR CMC	PCMC C.T	NIPW CMC	
1	4	78.2	78	63	0.	78.2	79.2	79.2	0.	96.2	11
1	2	13.6	13	11	0.	13.6	14.6	110.0	95.4	129.9	12
2	2	49.4	49	42	70.5	119.9	121.9	121.9	0.	144.2	14
5	4	119.5	119	109	0.	119.5	120.5	187.2	66.7	217.1	21
4	3	84.8	84	84	0.	84.8	85.8	187.2	101.4	220.0	22
6	4	139.8	139	134	67.3	207.1	209.1	209.1	0.	232.4	23
8	1	182.0	181	146	0.	182.0	183.0	183.0	0.	191.2	30
7	2	181.3	181	154	0.	181.3	182.3	302.3	120.0	320.7	32
9	1	213.0	212	200	0.	213.0	214.0	302.3	88.3	323.4	32
10	1	306.2	306	286	7.2	313.4	315.4	315.4	0.	333.4	33
11	1	318.1	318	312	3.3	321.4	323.4	323.4	0.	343.4	34

MS2BTC01 FILE - 6

C I S S

DM READ WRITE LOGS

NODE 5

NODE 6

TRAN#	R/W	CLS	TRAN#	R/W	CLS
1	R	2	1	R	4
3	W	4	3	W	4
1	W	2	1	W	2
2	R	2	1	W	2
2	W	2	1	R	3
8	R	1	1	P	4
5	W	4	1	W	1
4	W	3	1	W	3
6	W	4	1	W	4
7	P	2	1	P	4
8	W	1	1	W	4
9	R	1	1	W	2
7	W	2	1	W	1
9	W	1	1	W	1
10	P	1	1	W	1
10	W	1			
11	P	1			
11	W	1			

AD-A117 990

WEIZMANN INST OF SCIENCE REHOVOTH (ISRAEL) DEPT OF --ETC F/6 9/2
COMPARATIVE PREDICTION METHODOLOGY FOR DECENTRALIZED DDBMS.(U)
APR 82 M MELMAN

AFOSR-81-0147

UNCLASSIFIED

EOARD-TR-82-11

NL

2 OF 2
AD A
117990



END
DATE
FILMED
09:82
DTIC

S S

15.54.45 02- 5-82 PAGE- 1

E

RCMQ C.T	NIRW CMQ	LVW CMQ	WCPC Q.T	SERVICE TIME	COMPL TIME	RESP TIME	SYS TIME	NCDE	REJ
0.	96.2	110.0	13.8	41.9	86.2	8.0	41.9	4	
95.4	149.9	149.9	0.	117.5	119.9	106.3	117.5	2	
0.	144.2	144.2	0.	26.3	134.2	84.8	96.8	4	
66.7	217.1	217.1	0.	98.2	207.1	87.6	98.2	4	
101.4	220.0	220.0	0.	135.4	203.0	118.2	135.4	3	
0.	232.4	232.4	0.	26.9	222.4	82.6	94.2	4	
0.	191.2	302.3	111.1	124.4	190.2	8.2	124.4	1	
120.0	320.7	320.7	0.	147.2	310.7	129.4	147.2	2	
88.3	323.4	323.4	0.	116.1	313.4	100.4	116.1	1	
0.	333.4	333.4	0.	23.3	321.4	15.2	30.5	1	
0.	343.4	343.4	0.	26.0	332.1	14.0	29.3	1	

S

15.54.50 02- 5-82 PA

RUN NO. 2 : SIMPLE MODEL , STATISTICS ONLY
=====

INPUT FOR THE RUN
=====

GLOBAL PARAMETERS
=====

TIME UNITS	MILLISECONDS
NUMBER OF CLASSES IN THE SYSTEM	4
MAX NUM OF FRAGMENTS PER SET	C
AVERAGE LENGTH OF A CONTROL MESSAGE	0.50 (UNITS OF TIME)
BATCH SIZE	25
CONFIDENCE LEVEL	0.90

FLAGS
=====

TRACE FLAG	0
TRACE DRIVEN FLAG	0
LOG FLAG	0

PROTOCOL NUMBERS
=====

CLASS	1	2	3	4
PROTOCOL	1	2	3	1

CONFLICT CLASS MATRIX
=====

	1	2	3	4
1	0	0	0	0
2	1	0	1	0
3	0	0	0	1
4	0	0	0	0

NODE CHARACTERIZATION
=====

NODE ID	PROG TYPE	CLASS	TRAN INT ARR (EXPC)	SEED	AV PROG LENGTH (EXPC)	SEED	DATA ACCESS TIME (EXPC)	SEED	TIME CUT (FIX)	DLT PER FLG	READ TIME STAMP DELT
1	TM	1	100.0	4	2.0	5			100.0	1	.200
2	TM	2	100.0	6	2.0	7			100.0	1	.200
3	TM	3	100.0	8	2.0	9			100.0		
4	TM	4	100.0	10	2.0	11			100.0	1	.200
5	DM						3.0	2	100.0		
6	DM						3.0	3	100.0		

S S

18.36.28 02- 5-82 PAGE- 2

, STATISTICS ONLY
=====

OF TIME)

TIME CUT (FIX)	DLT PCR FLG	REAC TIME STAMP DELTA
100.0	1	.2000
100.0	1	.2000
100.0		
100.0	1	.2000
100.0		
100.0		

SYSTEM PERFORMANCE RESULTS

SIMULATION TIME = 100000.0 UNITS

TM NODE STATISTICS

NOD ID	NUM TRAN COMP	TRANS TM C. TIME	AVG # WAIT. TRANS	% CF REJEC TRANS	TM UTIL.	AVG RESPO TIME	RESPD TIME STC	B AV ST
1	974	14.070	.137	0.	.309	45.943	46.135	
2	729	38.455	.380	26.140	.573	109.104	47.364	
3	920	31.196	.321	10.593	.521	81.802	41.073	
4	943	6.871	.065	0.	.210	29.203	33.819	

DM NODE STATISTICS

NOD ID	READ CMQ TIME	WRITE CMQ TIME
5	38.581	7.970
6	26.893	1.847

GLOBAL STATISTICS

NUMBER OF TRANSACTIONS ARRIVED IN SYSTEM

NUMBER OF TRANSACTIONS PROCESSED (AFTER WRITES)

NUMBER OF TRANSACTIONS REJECTED

NUMBER OF MESSAGES SENT PER TRANSACTION , AVG (INC. NW)

AVG. SYSTEM TIME PER TRANSACTION

AVG. TRANSMISSION TIME PER TRANSACTION (NOT INC. NW)

AVG. RESPONSE TIME PER TRANSACTION

SPO PE	RESPC TIME STC	BATCH AVG STC	RELAT PREC1 IN %	CORR	NUM CF BATCHES
5.943	46.135	17.161	10.223	-.051	36
9.104	47.362	11.075	3.189	.020	29
1.802	41.073	12.012	4.135	-.187	36
9.203	33.819	10.313	9.802	-.054	37

3935
3562
367
8.9858
89.0241 UNITS
18.6173 UNITS
63.7898 UNITS

THE RUN NEEDED 59.98 CPU SECONDS

1 2

FILMED
9-8